Contribution to Solving Three-Dimensional Packing Problems

Labib Yousef

Thèse de Doctorat

Mention : Informatique
Spécialité : Recherche Opérationnelle et Optimisation

présentée à ***l'Ecole Doctorale en Sciences Technologie et Santé (ED 585)***

de l'Université de Picardie Jules Verne

présentée par

Labib YOUSEF

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

Contribution à la résolution des problèmes de placement en trois dimensions

Soutenue le 29-06-2017, après avis des rapporteurs, devant le jury d'examen :

M[me] Laetitia Jourdan	**Professeur, Université de Lille**	**(Rapporteur)**
M. Imed Kacem	**Professeur, Université de Lorraine**	**(Rapporteur)**
M[me] Myriam Sahnoun	**Maître de conférences, Université de Lorraine**	**(Examinateur)**
M. Said Hanafi	**Professeur, Université de Valenciennes**	**(Examinateur)**
M.Vassilis Zissimopoulos	**Professeur, Université Nationale et Kapodistrian d'Athènes**	**(Examinateur)**
M. Mhand Hifi	**Professeur, Université de Picardie Jules Verne**	**(Directeur)**

REMERCIEMENTS

Tout d'abord, je voudrais exprimer ma plus profonde gratitude à mon directeur de thèse, Professeur Mhand Hifi, sans lui ce travail n'aurait pas été possible. Je suis très reconnaissant envers lui de m'avoir guidé tout au long de la préparation de cette étude. Son suivi, son soutien régulier et ses idées inépuisables ont été extrêmement utiles dans mes recherches.

Je remercie profondément les Professeurs Imed Kacem [1] et Laetitia Jourdan [2] d'avoir accepté de rapporter mes travaux de recherche et l'intérêt qu'ils ont porté à mon travail. Je voudrais également remercier les Professeurs Vassilis Zissimopoulos [3], Said Hanafi [4] et Mme Myriam Sahnoun [5] d'avoir accepté de participer à mon jury de la thèse.

Je suis heureux d'avoir passé mes études de doctorat au laboratoire EPROAD. Je tiens à remercier tous les membres de ce laboratoire avec qui j'ai partagé beaucoup de discussions et de beaux moments.

Je voudrais aussi remercier l'Université de Damas par le Ministère de l'Enseignement Supérieur de la Syrie qui ont financé ces années de thèse de doctorat. Grâce à ce financement, j'ai pu accomplit mes travaux de recherche de bonnes conditions. Encore une fois merci.

Une gratitude et un amour particuliers vont à ma famille pour leur soutien infaillible. Je remercie mes parents pour leur amour permanent. Je remercie ma fille, Julia, pour son doux sourire. Enfin, je veux exprimer mon amour le plus profond et les remerciements à ma femme, Taghred, pour son soutien et de m'avoir supporté sur toute la durée de cette thèse.

[1] Professeur à l'Université de Lorraine
[2] Professeur à l'Université de Lille
[3] Professeur à l'Université nationale et Kapodistrienne d'Athènes
[4] Professeur à l'Université de Valenciennes
[5] Maître de Conférences à l'Université de Lorraine

Résumé

Titre : Contribution à la résolution des problèmes de placement en trois dimensions.

Les problèmes de découpe/placement interviennent dans de nombreux domaines industriels tels que le transport, la logistique et la production. Ils apparaissent soit en tant que problème principal, soit en tant que sous-problèmes de problèmes plus complexes.

Ce travail s'intéresse à la résolution approchée (heuristique et métaheuristique) de nombreuses variantes du problème de découpe/placement (Cutting & Packing, notée C&P). Le problème de placement de sphères dans un container parallélépipède ouvert représente la première variante du problème traitée dans cette thèse. Le placement de sphères dans un container parallélépipède fermé est la deuxième variante du problème traitée. Finalement, le placement de spheres dans un container sphérique, qui repreésente la troisième variante abordé dans cette thèse.

Pour ces variantes, nous proposons quatre méthodes de résolution. La première méthode s'appuie sur une recherche dichotomique et une recherche arborescente par faisceaux. Le but est de minimiser la longueur du container ouvert tout en plaçant l'ensemble des sphères disponibles.

La deuxième méthode peut être vue comme une amélioration de la première méthode pour résoudre la même variante du problème de placement. Elle s'appuie sur la recherche par faisceaux combinée à la recherche dichotomique et une nouvelle estimation de la borne inférieure pour ce problème. En effet, la notion d'estimation a été introduite afin d'explorer efficacement des espaces de recherche dans lesquelles la qualité des solutions est à privilégier.

La troisième méthode s'appuie sur la recherche à voisinage large combinée à une méthode d'optimisation continue. Le but, étant de maximiser la densité du placement dans un container fermé. Cette approche démarre d'une configuration quelconque et converge vers une solution réalisable en s'appuyant sur une recherche par voisinage large pour la diversification et en appliquant une méthode d'optimisation continue.

Finalement, nous proposons une méthode d'optimisation par essaims particulaires combinée avec une procédure d'optimisation continue pour résoudre le problème de placement de sphères identiques dans un container sphérique fermé ou un container de forme parallélépipède ouvert. La procédure d'optimisation continue est utilisée pour réparer les solutions non réalisables produites lors de la résolution.

Mots clés: dichotomie; essaims particulaires; (méta)heuristiques; optimisation; voisinages.

Abstract

Title: Contribution to Solving Three-Dimensional Packing Problems

Cutting and Packing (C&P) problems are encountered in numerous industrial domains such as transportation, logistics, reliability, and production. They appear either as standalone problems or as subproblems of more complex problems.

The goal of the thesis is to investigate the use of heuristics and meta-heuristics for solving variants of cutting and packing problems. Packing spheres into an open container represents the first variant of the problem. Packing spheres into a closed container is the second variant. Finally, packing spheres into a spherical container is the third variant studied in the thesis.

These variants are solved by using four solution methods. The first approach is based upon a dichotomous search and a truncated tree search (beam search). The goal is to determine the minimum length of the open container that contains all spheres without overlapping between all items.

The second approach can be viewed as a modified version of the first one, for solving the same variant of the problem, where a tree search (beam search) combined with the dichotomous search and the estimate of the lower bound is proposed. Herein, the lower bound is used in order to guide the search process more efficiently where primarily the quality of the solutions is preferred.

The third method is based upon the large neighborhood search combined with a continuous optimization algorithm for solving the problem of packing spheres into a close container. Starting from any configuration, the goal of the continuous optimization is to converge to a feasible solution whereas the large neighborhood search offers a diversification of the search space to enable convergence toward the solutions of best qualities.

Finally, the particle swarm optimization combined with a continuous optimization algorithm is proposed to tackling the (identical) sphere packing problem into different containers.

Keywords: dichotomous; heuristics; neighbors; optimization; particle swarm.

Contents

List of Figures

List of Tables

Introduction génerale

Les problèmes de découpe et de placement (Cutting & Packing: C&P) se présentent traditionnellement lorsqu'on cherche à extraire, découper ou placer, un ensemble prédéfini d'éléments à partir de grands objets tout en respectant une ou plusieurs contraintes. Les contraintes peuvent être liées aux chevauchements, à l'équilibre, au type de découpe, à la capacité ou au temps. L'optimisation concerne une fonction objectif qui peut dépendre des containers et des éléments sélectionnés à découper ou placer. Parfois l'objectif peut être mupltiple, par exemple, l'optimisation du placement, l'optimisation du profit associé à un placement, l'optimisation selon des critères, etc. Une contrainte de placement peut également dépendre de la position des éléments à placer dans un ou plusieurs containers.

Ces problèmes se retouvent dans de nombreux domaines industriels tels que le transport, la logistique et la production. Chacun de ces problèmes peut apparaitre soit en tant que problème principal, soit en tant que sous-problème d'un autre problème plus complexe à résoudre. En raison de l'importance des champs d'applications, ces problèmes se déclinent sous forme de variantes, par exemple:

- Placement ou découpe d'objets (circulaires, sphériques, rectangulaires, parallélépipèdes) dans un seul (plusieurs) container(s) fermé(s).
- Placement ou découpe d'objets (circulaires, sphériques, rectangulaires, parallélépipèdes) dans un seul (plusieurs) container(s) ouvert(s).
- Placement ou découpe d'objets irréguliers dans un (plusieurs) container(s).
- Placement d'objets spécifiques avec des contraintes spécifiques: arrangement d'objets sur des surfaces spécifiques en respectants des distances, des contraintes disjonctives, ...

Malgré l'avance technologique et scientifique, la résolution des problèmes de type C&P reste difficile. Des techniques de résolution basées sur des approches "efficaces" sont souvent proposées. Dans cette thèse, nous nous intéressons à la résolution des problèmes de placement de sphères par application de méthodes approchées (heuristiques et métaheuristiques). Un domaine d'application pratique est ciblé ici, en particulier, la problématique générale du placement de sphères dans un container ouvert, un container fermé et un container sphérique. Un tel problème se rencontre dans :

1. L'étude des structures de divers systèmes physiques, comme les milieux granulaires, les liquides, les cellules vivantes et les milieux aléatoires en chimie et en physique (cf. Torquato [71]) ;

2. Les applications médicales que l'on rencontre dans la modélisation de la problématique liée au traitement du cancer par radiochirurgie (cf. Wang [73], Sutou *et al.*[69]).

3. Le domaine de la communication et du stockage numérique (cf. Conway *et el.* [17], Claudio *et al.* [13]).

Nous nous sommes interessés dans cette thèse à la résolution heuristique et métaheuristique du problème de placement de sphères. Cette thèse est organisée en trois parties.

Dans une première partie, nous présenterons un cadre général de la problématique traitée. Nous présenterons le problème de placement de sphères qui fait partie des problèmes d'optimisation combinatoire/continue les plus connus de la littérature. En effet, cette problématique fait partie à la fois des problèmes d'optimisation combinatoire et des problèmes d'optimisation continue, car la structure d'une solution réalisable est de nature discrète alors que les positions des éléments sphériques à placer sont de nature continue.

Nous donnerons également une classification des problèmes de découpe/placement très utilisée dans la littérature et la position de notre problème dans cette classification. Par la suite, nous ferons un zoom sur les différentes variantes du problème de placement de sphères, à savoir, le placement de sphères dans un container ouvert, le placement de sphères dans un container fermé et enfin, le placement de sphères dans un container sphérique.

Nous présenterons, par la suite, quelques méthodes de résolution existantes que nous avons étudiées au cours de cette thèse. Nous terminerons ce chapitre par une conclusion qui résume le contenu du chapitre et explique la démarche suivie dans la suite de la thèse.

La deuxième partie contient nos contributions. Elle est divisée en quatre chapitres. Dans le premier chapitre, une méthode de recherche arborescente est proposée afin de résoudre le problème de placement dans un container ouvert. Le but est de minimiser la longueur de ce dernier tout en plaçant l'ensemble des sphères disponibles. La méthode s'appuie principalement sur la recherche arborescente dans laquel trois phases complémentaires sont combinées : (i) une phase de sélection selon un critère de choix local qui détermine une série de sous-espace, (ii) une recherche par faisceaux, qui explore quelques chemins prometteurs en visant à trouver les meilleurs chemins et (iii) une recherche dichotomique qui diversifie l'espace de recherche.

Dans le deuxième chapitre, une nouvelle version de la méthode proposée dans le précédent chapitre. Le but de cette méthode est d'améliorer la qualité des solutions tout en limitant le temps de calcul, qui parfois peut devenir exorbitant pour ce type de problème. En effet, la méthoe proposée s'appuie sur la recherche arborescente par faisceaux combinée à la recherche dichotomique et l'estimation d'une nouvelle borne inférieure. Ici, la notion d'estimation a été introduite afin d'explorer efficacement des espaces de recherche dans lesquelles la qualité des solutions est à privilégier.

Dans le troisième chapitre, une nouvelle méthode hybride combinant une recherche

à voisinage large et méthode d'optimisation continue est proposée. En partant d'une configuration quelconque, la méthode d'optimisation a pour but de converger vers une solution réalisable. La méthode à voisinage large propose, quant à elle, une diversification de l'espace de recherche afin de permettre une convergence vers des solutions de meilleures qualités. Notons aussi que cette méthode utilise une prodédure de type glouton auparavant proposée dans les premiers chapitres permettant la construction d'une solution de départ de qualité plus ou moins de qualité.

Enfin, l'optimisation par essaims particulaires combinée avec une procédure d'optimisation continue est proposée pour résoudre le problème du placement de sphères identique dans un container sphérique ouvert. La procédure d'optimisation continue est utilisée pour réparer l'inadmissibilité des solutions.

Dans la troisième partie, nous présenterons les perspectives et les directions de nos travaux futurs.

Chapter 1

Introduction aux problèmes de placement

Contents

Dans ce chapitre, nous présenterons un cadre général de la problématique traitée. Il s'agit du problème de placement de sphères, qui fait partie des problèmes d'optimisation combinatoire. Ces problèmes peuvent être aussi traités par des modèles non-linéaires à variables continues. Cette problématique appartient en effet à la fois aux problèmes d'optimisation combinatoire et aux problèmes d'optimisation continue puisque la structure d'une solution réalisable (admissible) est de nature discrète alors que les positions des objets à placer dans un (ou plusieurs) container(s) sont de nature continue.

Nous donnerons également une des classifications utilisée dans la littérature pour des problèmes dits de découpe et placement et, le positionnement des problématiques que nous avons traitées dans nos études. Par la suite, nous détaillerons les différentes variantes du problème de placement de sphères et leurs domaines d'application, c'est-à -dire le placement de sphères dans un container ouvert (Three-Dimensional Sphere Packing Problem : noté $3D - SPP$), le placement de sphères dans un container fermé (Three-Dimensional Knapsack Problem : noté $3D - KP$) et, enfin, le placement de sphères dans un container sphérique (Three-Dimensional Packing Spheres into a Sphere: noté $3D - PSS$).

1.1 Problème d'optimisation

Un problème d'optimisation peut être défini comme la recherche d'une valeur minimale ou maximale d'un objectif (fonction mathématique) donné(e) en respectant un certain nombre de contraintes également formulées sous forme mathématique (cf., Collette et Siarry [16]). Dans le cas d'un problème de minimisation, la formulation mathématique peut être représentée comme suit :

$$\min \quad f(\overrightarrow{x}) \forall \overrightarrow{x} \in S \tag{1.1}$$
$$g(\overrightarrow{x}) \leq 0 \tag{1.2}$$
$$h(\overrightarrow{x}) = 0 \tag{1.3}$$

En d'autres termes, le problème d'optimisation consiste à trouver la meilleure solution $\overrightarrow{x}^*$ parmi l'ensemble des solutions possibles (et existantes) dans l'espace de recherche, tout en respectant l'ensemble des contraintes du problème. Mathématiquement, une solution $\overrightarrow{x} = \{x_1, x_2, \therefore, x_n\}$ est un vecteur contenant n variables de décision (avec $n > 0$). Si ces variables sont des nombres réels, le problème d'optimisation est dit continu $\overrightarrow{x} \in \mathbb{R}^n$ et $S \subseteq \mathbb{R}^n$. Cependant, si les variables sont discrètes, le problème d'optimisation est dit discret (ou parfois combinatoire) $\overrightarrow{x} \in \mathbb{Z}^n$ et $S \subseteq \mathbb{Z}^n$.

Résoudre un problème d'optimisation revient à trouver une solution réalisable (qui respecte l'ensemble des contraintes). Selon l'espace de recherche ou la partie de l'espace de recherche qu'on souhaite examiner, nous distinguons deux solutions réalisables : (i) le minimum / maximum local et, (ii) le minimum / maximum global.

Definition 1. *Une solution $\overrightarrow{x}^* \in S$ est considérée comme un minimum / maximum global de la fonction f s'il n'existe pas* ***dans l'espace de recherche*** *S une autre solution de meilleure valeur (évaluation ou qualité), c'est-à -dire :*

$$\forall \overrightarrow{x} \in S = \begin{cases} f(\overrightarrow{x}^*) < f(\overrightarrow{x}) & \textit{cas d'une minimisation} \\ f(\overrightarrow{x}^*) > f(\overrightarrow{x}) & \textit{cas d'une maximisation} \end{cases}$$

Definition 2. *Une solution $\overrightarrow{x}^*$ est considérée comme un minimum / maximum local de la fonction f s'il n'existe pas dans* ***ses voisinages*** *$V(\overrightarrow{x}^*)$ une autre solution de meilleure valeur (évaluation ou qualité), c'est-à -dire :*

$$\forall \overrightarrow{x} \in V(\overrightarrow{x}^*), \overrightarrow{x} \neq \overrightarrow{x}^* \begin{cases} f(\overrightarrow{x}^*) \leq f(\overrightarrow{x}) & \textit{cas d'une minimisation} \\ f(\overrightarrow{x}^*) \geq f(\overrightarrow{x}) & \textit{cas d'une maximisation} \end{cases}$$

La figure 1.1 illustre un minimum (maximum) global et un autre minimum (maximum) local pour un problème d'optimisation avec un objectif.

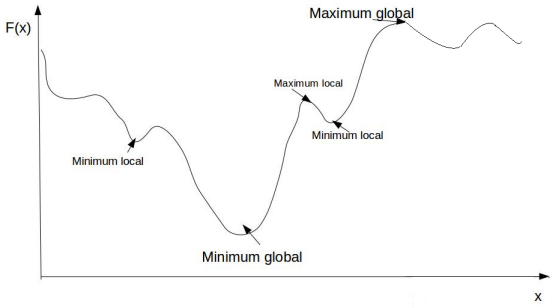


Figure 1.1: Les différents minima

1.2 Les problèmes de découpe / placement

Les problèmes de découpe et de placement (Cutting & Packing : noté C&P) se présentent traditionnellement lorsqu'on cherche à extraire (découper) ou à placer (positionner) un ensemble prédéfini d'objets à partir de (dans de) grands objets tout en respectant une ou plusieurs contraintes spécifiques (par exemple, la contrainte liée au chevauchement entre les objets, la découpe de type guillotine, l'équilibrage lié aux positionnements des objets, la capacité des objets, la contrainte liée au temps, etc.) et en optimisant une certaine fonction objectif qui peut dépendre des containers et des objets sélectionnés ou positionnés dans le ou les containers (cf., Dyckhoff *et al.* [26, 21], Wäscher *et al.* [74]).

Ces problèmes font partie des problèmes d'optimisation combinatoire les plus connus dans la littérature. Ils ont fait l'objet de nombreuses études scientifiques avec un nombre de publications croissant depuis de nombreuses années et ils possèdent de nombreuses applications pratiques importantes en industrie (découpe de matières premières), logistique de transport (cargaison, véhicules, camions, palettes, etc.), stockage (problème d'allocation de mémoire), finance (gestion de portefeuille).

Face à l'importance de ces problématiques et au nombre important de sous problématiques liées à la découpe et le placement, Wäscher *et al.* [74] ont proposé une typologie sur les problèmes de la famille C&P en tenant compte des cinq critères les plus répondus :

1. La dimension du problème : il s'agit des dimensions des objets (items) dans un problème traité.

2. L'objectif : lorsque l'ensemble des containers disponibles ne suffit pas à contenir tous les objets, alors on s'intéresse à maximiser le placement d'un ensemble d'objets dans un ensemble de containers disponibles. Dans le cas contraire, l'objectif revient à minimiser le nombre de containers à utiliser (dans le cas d'un seul container, il revient à minimiser la dimension du container qui peut varier).

3. Les objets à placer : la forme, la taille et la distribution des objets à placer. Trois cas

peuvent être distingués : (a) tous les objets ont la même taille, (b) on ne considère qu'un petit nombre de classes d'objets de tailles différentes et, (c) le nombre d'objets est très important et ils n'ont pas la même taille.

4. Les containers à utiliser : la nature, la taille et le nombre de containers à utiliser. Deux cas peuvent être distingués : (a) un seul container est disponible et ses dimensions peuvent être fixes ou variables et, (b) plusieurs containers de dimensions fixes de même taille ou de tailles différentes.

5. La forme des objets : les objets peuvent être représentés par des cercles, des rectangles, des cylindres, des sphères, des cubes, etc. De même, le type des objets à placer (ou à découper) est du même type que les containers.

Finalement, Wäscher *et al.* [74] ont proposé six classes de problèmes de la famille C&P. Elles sont catégorisées selon les critères 2 et 3 précédemment décrits (cf, Figure 1.2) :

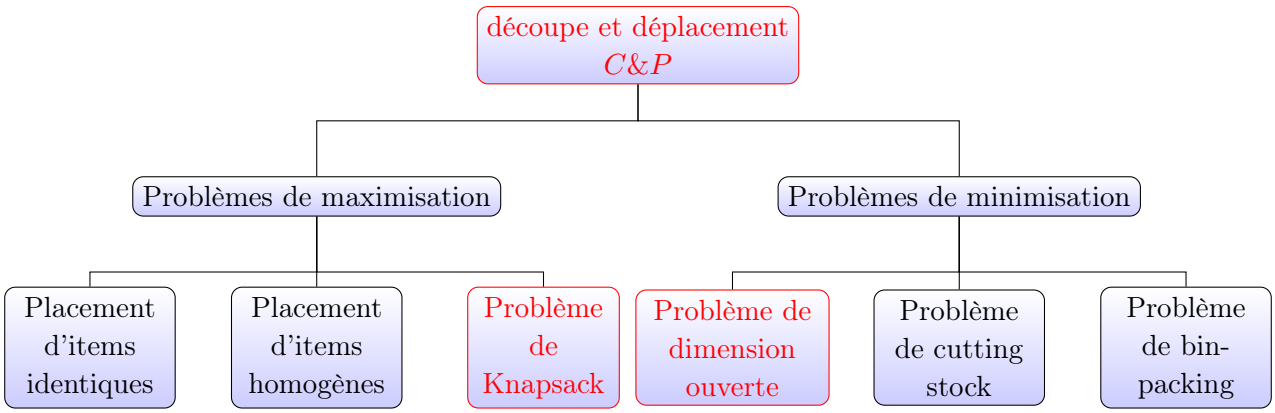


Figure 1.2: Principales classes de problèmes de la famille C&P

Dans cette thèse, nous nous intéressons aux deux classes suivantes :

- Classe des problèmes avec une dimension du container ouverte (Open Dimension Problem : ODP) : on dispose d'un seul container ouvert ou d'un container sphérique fictif (à déterminer), alors que les objets à placer sont sphériques, où chaque objet est caractérisé par son rayon (ces objets peuvent être identiques ou non-identiques).

- Classe des problèmes de type sac à dos (Single Knapsack Problem : SKP) : le container (qui peut être sphérique) est fermé (de dimensions fixes) et les objets à placer sont sphériques non-identiques.

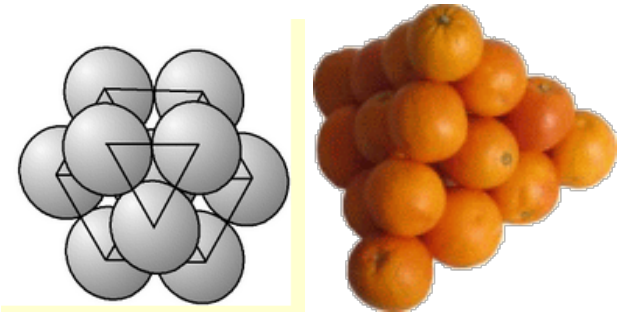
1.3 Le problème de placement de sphères

1.3.1 Des applications

Les problèmes de placement de sphères sont connus depuis 1610, lorsque la question la plus célèbre en mathématiques a été posée (cf., Conway *et al.* [17]) :

"Quelle est la densité maximale d'un empilement compact de sphères identiques dans l'espace ?"

En 1611, Kepler [17], astronome et mathématicien Allemand, émit la conjoncture (sans preuve) que la densité maximale était d'environ $\frac{\pi}{3\sqrt{2}} \simeq 74.048\%$. Cette densité est atteinte pour un placement cubique à faces centrées. Cette situation se retrouve par exemple sur les étals de magasins de fruits et de légumes, comme l'illustre la Figure 1.3.



(a) (b)

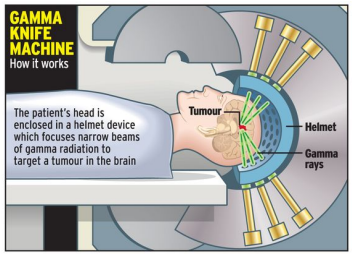
Figure 1.3: Structure du placement correspondant à la conjecture de Kepler

Le problème de "Kissing Number", qui consiste à déterminer le plus grand nombre de sphères identiques pouvant être rangées autour d'une sphère du même type, était un sujet de désaccord entre Newton (physicien, philosophe, astronome et mathématicien Anglais) et Gregory (mathématicien et astronome) en 1690. Newton pensait qu'on ne pouvait ranger que 12 sphères autour d'une sphère de même rayon, tandis que Gregory disait qu'on pouvait en mettre 13 (cf., Conway *et al.* [17], Kucherenko *et al.* [47], Zong [79]). En 1956, Leech [49] prouva que ce nombre était de 12. Hilbert (un mathématicien Allemand) inclut en 1900 (cf. Lagarias [48]) la conjoncture de Kepler dans sa célèbre liste des 23 grands problèmes non résolus en mathématique (cf., Derbyshire [20]). En 1998, Hales proposa une preuve de la conjoncture de Kepler à l'aide de calculs informatiques. Sa démonstration comprenait 300 pages de texte et 40 000 lignes de code (programme). En 2014, la conjoncture de Kepler fut officiellement démontrée.

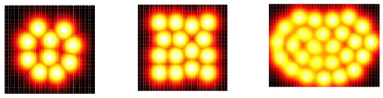
En pratique, les applications de placement de sphères sont très nombreuses, par exemple:

En médecine : dans le domaine du traitement des cancers, le gamma knife (cf., Figure 1.4(a)) est l'une des techniques de radio-chirurgie les plus efficaces. Il s'agit d'envoyer des doses plus élevées de rayonnement (connus sous le nom de "Shots") sur la région cancéreuse (cf., Figure 1.4(b)) afin de provoquer des réactions permettant des évolutions vers la guérison du patient.

(a) Gamma knife



(b) Traitement des cancers par Gamma knife



(c) Modélisation des shots et la région cancéreuse

Figure 1.4: Illustrations du traitement par Gamma knife

Un traitement optimal par le Gamma knife doit vérifier les critères suivants (cf., Wang [73], Sutou *et al.* [69], Li *et al.* [50]) :

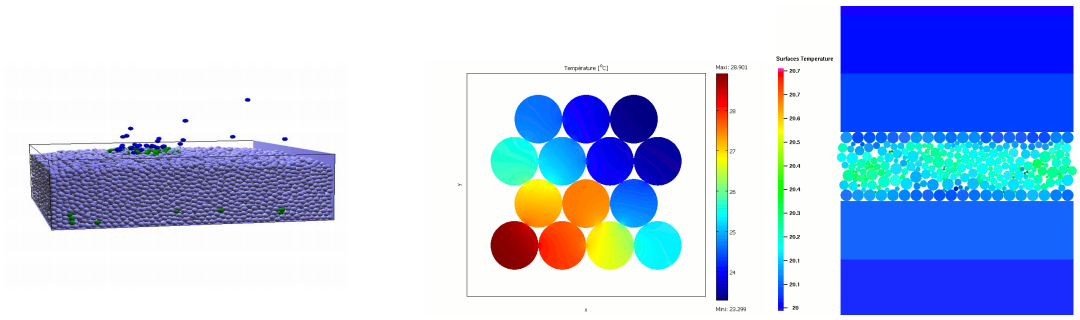
1. Le rayonnement ne doit pas dépasser les bords de la région ciblée afin de ne pas endommager les cellules saines.
2. Les chevauchements entre les rayonnements sont interdits afin d'éviter un surdosage.
3. Le meilleur traitement est celui qui génère le plus de densité et qui évite un sous-dosage ou une distribution non uniforme.

Du point de vue de la modélisation (cf., Figure 1.4(c)), les rayonnements peuvent être considérés comme des sphères, identiques ou non-identiques, et la région cancéreuse peut être représentée par une forme géométrique, régulière ou irrégulière.

En parallèle, Sutou *et al.* [69] et Li *et al.* [50], ont proposé le placement de sphères non-identiques dans un polytope pour modéliser le traitement précédemment décrit tandis que Wang [73] a proposé le placement de sphères non-identiques dans une région irrégulière.

Science des matériaux et biologie : le placement de sphères a été utilisé comme modèle pour représenter les milieux granulaires et la structure des lipides, des protéines

et des matériaux vitreux afin d'étudier quelques propriétés concernant la conductivité thermique, la conductivité électrique, les mouvements de fluides, la distribution des contraintes et d'autres propriétés mécaniques des granulaires, des cellules vivantes, etc. (cf., Williams *et al.* [75], Larrard *et al.* [18], Cheng *et al.* [12]). Dans cette partie, nous présentons le modèle de placement de sphères proposé par Nguyen *et al.* [59, 60] pour étudier le problème de transfert thermique dans les milieux granulaires.



(a) Milieu granulaire

(b) Transferts thermiques par conduction entre les particules

(c) Transferts thermiques fluide-particules

Figure 1.5: Transferts thermiques en milieux granulaires

En effet, les milieux granulaires, comme le montre la Figure 1.5(a), sont généralement des milieux poreux diphasiques (ils contiennent des particules et des pores). Les particules (ou grains) sont modélisées sous forme de sphères identiques (ou non-identiques) stockées dans une matrice (un container). Les espaces vides autour des particules sont considérés comme des pores, comme le montre la Figure 1.5(b). L'augmentation de la conductivité thermique est due à une réduction des dimensions des pores ainsi qu'à une augmentation de l'espace entre les particules. Par ailleurs, Nguyen *et al.* [61] ont utilisé cette modélisation pour étudier la dissipation de l'énergie par frottement sur l'interface d'un système de freinage qui joue un rôle important dans la sécurité du transport automobile ou ferroviaire. Le modèle proposé est constitué de deux couches solides et d'une couche intermédiaire granulaire. Cette dernière a été modélisée comme un placement de sphères (ou cercles) dans un container, comme l'illustre la Figure 1.5(c).

En écologie : la modélisation de la structure complexe des forêts tropicales est l'un des grands défis. Une des questions dignes d'intérêt est la distribution des différentes tailles d'arbres, ce qui est particulièrement important pour estimer la biomasse. Taubert *et al.* [70] ont proposé un modèle reposant sur le placement de sphères pour étudier et présenter la structure des forêts tropicales (cf., Figure 1.6).

Communication et stockage numérique : la téléphonie, les réseaux informatiques, la correction d'erreurs de translation et de stockage des données, sont considérés comme

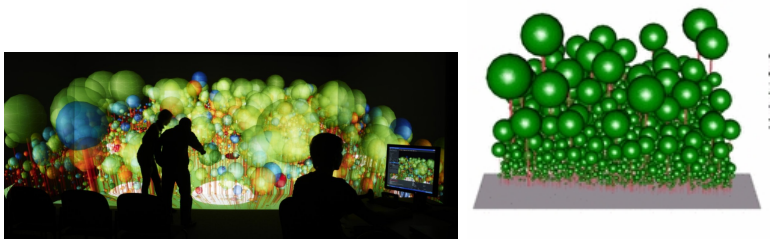


Figure 1.6: Modélisation de la structure complexe des forêts tropicales

des problèmes de placement de sphères sans chevauchement où l'on cherche à maximiser la densité dans une dimension donnée, comme l'expliquent Conway *et al.* [17] et Claudio *et al.* [13]. Il est à noter que d'autres applications sont présentées par Hifi *et al.* [31], où un ensemble de problématiques ont été abordées, modélisées et résolues par applications de diverses méthodes.

1.3.2 Quelques problèmes de placement de sphères

Dans cette thèse, nous nous intéressons à la résolution de trois variantes du problème de placement de sphères : le placement dans (i) un container ouvert, (ii) un container fermé et (iii) un container sphérique.

1.3.2.1 Le problème de placement dans un container ouvert

Une instance du problème de placement dans un container ouvert est représentée par un ensemble N de n objets (sphères), où chacun des objets est caractérisé par son rayon r_i (identique/non-identique) et un container ouvert, noté B de dimensions (H, W, ∞), dont H est sa largeur, W sa profondeur et $L = \infty$ sa longueur variable (cf., Figure 1.7). Le but du problème est de placer sans chevauchement tous les objets (sphères) dans le container de sorte à minimiser la longueur du container L.

Trouver une solution réalisable pour ce problème consiste donc à trouver les valeurs du couple (X, L), où $X = \{x_i, y_i, z_i, ..., x_n, y_n, z_n\}$ est un vecteur représentant les positions des n sphères placées dans le container B de longueur L (la Figure 1.8(a) illustre une solution réalisable associée à une instance du problème).

Dans ce qui suit, ce problème sera noté $3D - SPP$ (Three-Dimensional Strip Packing Problem) en suivant la même représentation de Kubach *et al.* [45, 46]). Le problème $3D - SPP$ peut être représenté par le modèle ci-dessous décrit (en supposant que l'origine se situe au point $(0, 0, 0)$) :

$$\min \quad L \tag{1.4}$$

$$\delta_{(S_i,S_j)} \geq (r_{S_i} + r_{S_j}) \quad \forall (i,j) \in N^2,\ i < j \tag{1.5}$$

$$x_{S_i} \leq L - r_{S_i} \qquad \forall i \in N \tag{1.6}$$

$$y_{S_i} \leq H - r_{S_i} \qquad \forall i \in N \tag{1.7}$$

$$z_{S_i} \leq W - r_{S_i} \qquad \forall i \in N \tag{1.8}$$

$$x_{S_i} \geq r_{S_i} \qquad \forall i \in N \tag{1.9}$$

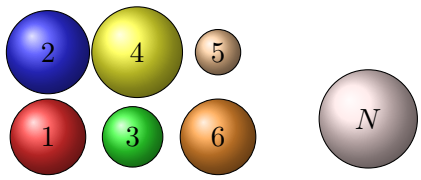
$$y_{S_i} \geq r_{S_i} \qquad \forall i \in N \tag{1.10}$$

$$z_{S_i} \geq r_{S_i} \qquad \forall i \in N \tag{1.11}$$

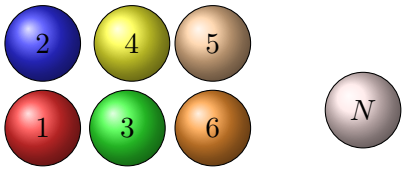
$$\underline{L} \leq L \leq \overline{L} \tag{1.12}$$

La contrainte 1.5 vérifie si deux sphères différentes S_i et S_j ne se chevauchent pas. La mesure $\delta_{(S_i,S_j)}$ représente la distance Euclidienne entre deux sphères.

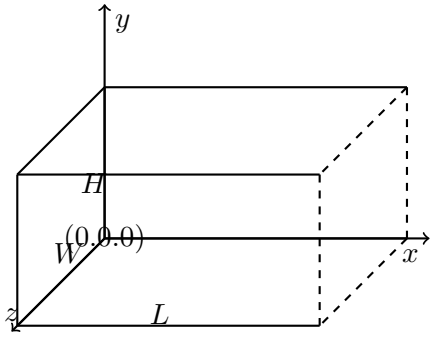
$$\delta_{(S_i,S_j)} = \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2} \tag{1.13}$$



(a) Sphères de rayons non identiques



(b) Sphères de rayons identiques



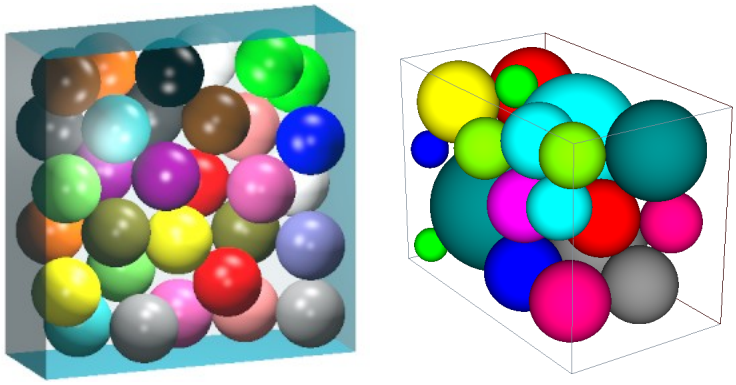
(c) Un container ouvert noté B

Figure 1.7: Une instance de $3D - SPP$

Les contraintes de 1.6 à 1.11 assurent que les sphères ne débordent pas du container. La représentation $(x_{S_i}, y_{S_i}, z_{S_i})$ représente la position de la sphère i dans le container B de longueur L. Quant à la contrainte 1.12, elle représente un encadrement de l'optimum par une borne inférieure et une autre borne supérieure. Une borne inférieure $\underline{L}$ peut être fournie de façon triviale, par exemple en posant:

$$\underline{L} = \frac{4 \times \pi \sum_{i=1}^{N} r^3}{3 \times W \times H}.$$

Par ailleurs, la borne supérieure $\overline{L}$ peut prendre au départ soit une valeur triviale comme $\overline{L} = 2 \times \underline{L}$, soit être calculée par un algorithme rapide de type glouton.



(a) Sphères de rayon $r_i = 1$ dans un container ouvert

(b) Sphères de rayons différents dans un container ouvert

Figure 1.8: Structures des solutions réalisables (de placement) dans un container ouvert B

Particularités du problème $3D - SPP$

À partir de la formulation du problème $3D - SPP$, nous distinguons les point suivants:

- Le nombre total de variables dans une instance de ce problème est de l'ordre de $3n + 1$.
- La plupart des contraintes sont non linéaires (quadratiques); nous avons donc $n \times (\frac{n-1}{2})$ contraintes de ce type. Nous avons aussi $6 \times n$ contraintes linéaires. Le nombre total de contraintes est de l'ordre de $6n + (N \times (\frac{n-1}{2}))$.
- La fonction objectif 1.4, qui représente la longueur minimum du container à déterminer, est linéaire. La variable L apparait uniquement dans la contrainte 1.6. Supposons que nous ayons une solution du problème $3D - SPP$, où toutes les sphères disponibles soient positionnées sans chevauchement dans le container B. Alors la variable L se calcule comme suit :

$$L = \max\{x_i + r_i\} \qquad \forall i \in N \tag{1.14}$$

- La fonction de pénalité, qui mesure la réalisabilité d'une solution donnée (X, L), se calcule à partir des quantités / valeurs de chevauchement entre les sphères ou entre les sphères et les dimensions (fixes) du container.

Le chevauchement entre deux sphères se calcule comme suit :

$$O_{i,j} = max\{0, r_{S_i} + r_{S_j} - \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2}\}. \quad (1.15)$$

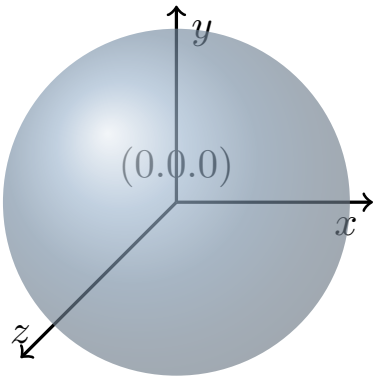
Par ailleurs, les chevauchements entre une sphère i et le container B sur les trois axes se calculent comme suit :

$$O_{i,X} = max\{0, r_{S_i} + |x_{S_i}| - 0.5L\} \qquad \forall i \in N \quad (1.16)$$
$$O_{i,Y} = max\{0, r_{S_i} + |y_{S_i}| - 0.5H\} \qquad \forall i \in N \quad (1.17)$$
$$O_{i,Z} = max\{0, r_{S_i} + |y_{S_i}| - 0.5W\} \qquad \forall i \in N \quad (1.18)$$

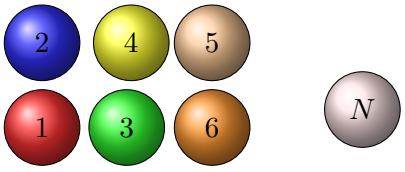
Enfin, la fonction de pénalité $E(X, L)$ se calcule en ajoutant les valeurs caractérisées par tous les chevauchements:

$$E(X, L) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} O_{i,j}^2 + \sum_{i=1}^{N} (O_{i,x} + O_{i,y} + O_{i,z}) \quad (1.19)$$

Une solution (X, L) est considérée comme solution réalisable *si et seulement si* $E(X, L) = 0$.



(a) Un container sphérique S



(b) Un ensemble de sphères de rayons identiques

Figure 1.9: Une instance du problème $3D - PSS$

1.3.2.2 Le problème de placement dans un container sphérique

Une instance du problème de placement dans un container sphérique est représentée par un ensemble N de n objets (sphères), où chacun des objets est caractérisé par son rayon ($r = 1$), et un container sphérique de rayon illimité ($R = \infty$) (cf., Figure 1.9). Le but du problème est de placer, sans chevauchement, tous les objets (sphéres) disponibles dans le container de sorte à minimiser le rayon R du container sphérique.

Ce problème, que l'on note $3D - PSS$ (Three-Dimensional Packing Smallest Sphere), a été étudié dans M'Hallah *et al.* [54]). La figure 1.10 illustre une solution réalisable du problème $3D - PSS$.

Figure 1.10: Structure d'une solution admissible de placement dans un container sphérique

D'une façon formelle, le problème $3D - PSS$ peut être représenté comme suit (cf., M'Hallah *et al.* [54]) :

$$\min \quad R \tag{1.20}$$

$$\delta_{(S_i,S_j)} \geq 2 \quad \forall (i,j) \in N^2,\ i < j \tag{1.21}$$

$$\delta_{(S_i,S)} \leq R - 1 \quad \forall i \in N \tag{1.22}$$

La contrainte 1.21 vérifie si deux sphères ne se chevauchent pas; dans ce cas, la distance Euclidienne $\delta_{(S_i,S_j)}$ entre les centres de deux sphères doit être supérieure à la somme de leurs rayons. Quant à la contrainte 1.22, elle assure que la sphère i est entièrement placée à l'intérieur du container sphérique (dit aussi la sphère fictive), où $\delta_{(S_i,S)}$ représente la distance Euclidienne entre le centre de l'objet (sphère) i et le centre du container sphérique. Nous distinguons les deux cas suivants :

(i) L'origine se situe au milieu de la grande sphère R. Alors,

$$\delta_{(S_i,S)} = \sqrt{x_{S_i}^2 + y_{S_i}^2 + z_{S_i}^2} \tag{1.23}$$

(ii) L'origine ne se situe pas au milieu de la grande sphère R. Alors :

$$\delta_{(S_i,S)} = \sqrt{(x_{S_i} - x_S)^2 + (y_{S_i} - y_S)^2 + (z_{S_i} - z_S)^2} \tag{1.24}$$

Particularités du problème $3D - PSS$

- Le nombre total de variables dans une instance de ce problème est de l'ordre de $3n + 1$.

- La plupart des contraintes sont non linéaires (quadratiques) ; nous avons donc $n + n \times (\frac{n-1}{2})$ contraintes de ce type.
- La fonction objectif 1.20 représente le rayon de la grande sphère (container sphérique) à optimiser (que nous cherchons à minimiser). Nous constatons que ce rayon n'apparait que dans la contrainte 1.22. Maintenant, supposons que nous ayons une solution (X, R) de 3D-PSS, où toutes les sphères disponibles soient positionnées sans chevauchement dans la grande sphère S. Alors le rayon R se calcule comme suit :

$$R = max\{\sqrt{x_{s_i}^2 + y_{s_i}^2 + z_{s_i}^2} + 1\} \qquad \forall i \in N \qquad (1.25)$$

- La fonction de pénalité $E(X, R)$, pour une solution (X, R) donnée, se calcule à partir des chevauchements comme suit :
  - le chevauchement entre deux sphères se calcule comme suit :

$$O_{i,j} = max\{0, 2 - \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2}\} \qquad (1.26)$$

  - le chevauchement entre une sphère et la grande sphère se calcule comme suit :

$$O_{i,S} = max\{0, R - \sqrt{x_{S_i}^2 + y_{S_i}^2 + z_{S_i}^2} + 1\}. \qquad (1.27)$$

  - La fonction de pénalité $E(X, R)$ se calcule finalement comme suit :

$$E(X, R) = \sum_{i=1}^{N} \sum_{j=0}^{N} O_{i,j}^2 \qquad (1.28)$$

Notons qu'une solution (X, R) est dite réalisable **si et seulement si** $E(X, R) = 0$.

1.3.2.3 Le problème de placement dans un container fermé

Une instance de placement dans un container fermé est représentée par un ensemble N de n objets (sphères), où chacune des sphères est caractérisée par son rayon r_i, et un container fermé (noté P) représenté par ses dimensions (L, W, H). Souvent, le container P n'est pas assez grand pour accueillir toutes les sphères de N. L'objectif est donc de maximiser la densité (notée D_{max}) en plaçant sans chevauchement un sous-ensemble $\overline{N}$ de N. À noter que si $\overline{N} = N$, alors nous avons une solution optimale pour cette instance du problème.

Notons aussi que, dans une version plus générale, chacune des sphères peut être représentée par un poids (ou un score / profit). Le but, dans ce cas, est alors de maximiser le poids des sphères à placer. Une solution réalisable du problème est illustrée par la Figure 1.11.

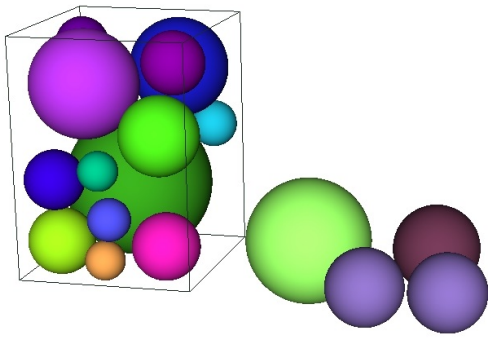


Figure 1.11: Une solutions réalisable contenant un nombre de sphéres non-identiques dans un container fermé

Ce problème peut être considéré comme une variante du problème du sac à dos (noté $3D-KP$: Three-Dimensional Knapsack Problem), comme il a été étudié dans Kubach *et al.* [45, 46]. Il peut aussi se modéliser sous la forme suivante (le modèle ci-dessous considère que le repère se situe au point (Bas, Derrière, Gauche) de coordonnées $(0, 0, 0)$) :

$$\max \quad \alpha \sum_{i \in N} r_i^3 \xi_i \tag{1.29}$$

$$\text{s.t.} \quad \xi_i r_i \leq x_i \leq \xi_i (L - r_i), \ i \in N \tag{1.30}$$

$$\xi_i r_i \leq y_i \leq \xi_i (H - r_i), \ i \in N \tag{1.31}$$

$$\xi_i r_i \leq z_i \leq \xi_i (W - r_i), \ i \in N \tag{1.32}$$

$$\xi_i \xi_j (r_i + r_j) \leq \sqrt{(x_i - x_j) + (y_i - y_j) + (z_i - z_j)}, \ (i, j) \in N^2, i < j \tag{1.33}$$

$$\xi_i \in \{0, 1\}, \ i \in N \tag{1.34}$$

$$\alpha = (4\pi)/(3LHW) \tag{1.35}$$

Particularités du problème $3D - KP$

En suivant le même raisonnement utilisé dans les sections précédentes (pour les deux autres variantes du problème), nous pouvons analyser la complexité spacial du modèle donné.

- Le nombre total de variables dans une instance donnée du problème $3D - KP$ est de $3n + 1$. La plupart des contraintes sont non linéaires (quadratiques) ; nous avons $n \times (\frac{n-1}{2})$ contraintes non linéaires. Nous avons aussi $6 \times n$ contraintes linéaires. Le nombre total de contraintes est de l'ordre de $6n + (N \times (\frac{n-1}{2}))$.

- Chaque contrainte est liée à une variable de décision ξ_i telle que :

$$\xi_i = \begin{cases} 1 & \text{la sphère } i \text{ est placée dans le container } P \\ 0 & \text{sinon} \end{cases}$$

- La fonction objectif (1.29) représente la densité D d'une solution réalisable. Cette dernière se définit comme la proportion du volume de P occupée par les sphères à l'intérieur où

$$\alpha = (4\pi)/(3LHW) \tag{1.36}$$

- La fonction de pénalité $E(X, P)$ et les chevauchements se calculent comme dans le cas du placement dans une bande (équations de 1.15 à 1.19).

Remarques

• Les différentes variantes du problème de placement de shpères sont NP-difficiles et le nombre de minima locaux est largement supérieur à $N!$ (cf., Stoyan *et al.* [67], Kubach *et al.* [46]). Il n'existe donc pas à ce jour d'algorithme qui puisse résoudre à l'optimum ce problème en un temps polynomial. Bien évidement, notre but n'est pas de s'étaler sur l'étude de la complexité algorithmique pour ces problèmes, puisque nous nous intéressons principalement à la résolution heuristique par des méthodes expérimentales.

• La classification de cette problématique "combinatoire ou continue" est floue (cf., Hifi *et al.* [31]). En effet, elle appartient à la fois aux problèmes d'optimisation combinatoire et aux problèmes d'optimisation continue, car la structure d'une solution réalisable est de nature discrète, alors que les positions des objets sphériques à placer dans un container sont de nature continue.

Dans ce cas, des méthodes peuvent être appliquées en ne gardant que l'aspect combinatoire même si le positionnement des objets reste continue. Dans d'autres cas, des méthodes d'optimisation en continue sont appliquées et l'aspect combinatoire peut être négligé puisque, par exemple, des méthodes de descentes s'intéressent principalement à l'aspect continue des variables (cf. Hifi *et al.* [31]).

1.4 Conclusion

Dans ce chapitre, nous avons décrit le cadre général de la problématique traitée dans cette thèse. Nous avons donné une classification utilisée dans la littérature sur les problèmes de découpe et placement et, la position dans cette classification des variantes étudiées dans cette thèse. Nous avons également présenté différentes variantes du problème de placement de sphères ainsi que certains domaines d'application, en l'occurrence le placement de

sphères dans un container ouvert, dans un container fermé et, enfin, dans un container sphérique.

L'étude bibliographique que nous avons menée en début de thèse nous a permis dans un premier temps d'apprécier la diversité des variantes des problèmes de découpe et de placement et, en particulier, le problème de placement de sphères. D'autre part, cette étude bibliographique nous a également permis d'étudier et de proposer plusieurs approches de résolution.

Chapter 2

Introduction to packing problems

Contents

In this chapter, we will present a general framework of the sphere packing problem, which belongs to combinatorial optimization problems. These problems can also be modeled as continuous non-linear problems. Indeed, they are at the same time combinatorial optimization problems as well as continuous optimization problems, since the structure of a feasible (admissible) solution is discrete whereas the positions of the objects to be placed in one (or several) container(s) are continuous.

We will also give one of the most utilized classifications of the literature for problems called cutting/packing problems and the position of our problem in this classification. Then, we will specify the different variants of the sphere packing problem and their application fields, especially the three variants studied in this thesis: sphere packing in an open container (Three-Dimensional Sphere Packing Problem, $3D - SPP$), sphere packing into a closed container (Three-Dimensional Knapsack Problem, $3D - KP$) and, finally, sphere packing into a spherical container (Three-Dimensional Packing Spheres into a Sphere, $3D - PSS$).

2.1 Optimization problem

An optimization problem can be defined as the search of a minimal or maximal value of a given objective (mathematical function) while following a certain number of constraints (cf. Collette and Siarry [16]). In the case of a minimization problem, the mathematical formulation can be represented as follows:

$$\min \quad f(\overrightarrow{x}) \forall \overrightarrow{x} \in S \tag{2.1}$$
$$g(\overrightarrow{x}) \leq 0 \tag{2.2}$$
$$h(\overrightarrow{x}) = 0 \tag{2.3}$$

In other terms, the optimization problem consists in finding the best solution $\overrightarrow{x}^*$ among the set of all possible solutions in the search space while respecting the problem's constraints. Mathematically, a solution $\overrightarrow{x} = \{x_1, x_2, \therefore, x_n\}$ is a vector which contains n decision variables (with $n > 0$). If these variables are real numbers, the optimization problem is said to be continuous $\overrightarrow{x} \in \mathbb{R}^n$ and $S \subseteq \mathbb{R}^n$. However, if the variables are discrete, the optimization problem is said to be discrete (or sometimes combinatorial) $\overrightarrow{x} \in \mathbb{Z}^n$ and $S \subseteq \mathbb{Z}^n$.

Solving an optimization problem is equivalent to finding a feasible solution (which respects all the constraints). According to the search space or the part of the search space that we wish to explore, we distinguish two feasible solutions: (i) the local minimum/ maximum and (ii) the global minimum/maximum.

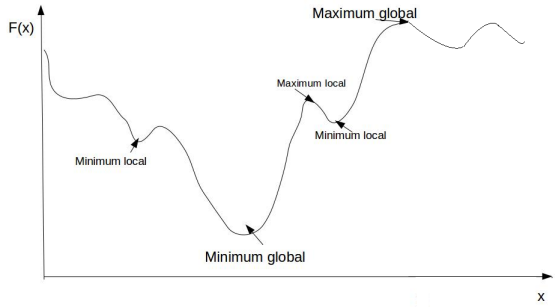


Figure 2.1: The different mimima

Definition 3. *A solution $\overrightarrow{x}^* \in S$ is considered a global minimum / maximum of function f if there is no better solution* ***in the search space*** *S, that is:*

$$\forall \overrightarrow{x} \in S = \begin{cases} f(\overrightarrow{x}^*) < f(\overrightarrow{x}) & \textit{minimization case} \\ f(\overrightarrow{x}^*) > f(\overrightarrow{x}) & \textit{maximization case} \end{cases}$$

Definition 4. *A solution $\overrightarrow{x}^*$ is considered as a local minimum / maximum of function f if there is no other better solution in its* ***neighbourhoods*** *$V(\overrightarrow{x}^*)$, that is:*

$$\forall \overrightarrow{x} \in V(\overrightarrow{x}^*), \overrightarrow{x} \neq \overrightarrow{x}^* \begin{cases} f(\overrightarrow{x}^*) \leq f(\overrightarrow{x}) & \textit{in the case of a minimization} \\ f(\overrightarrow{x}^*) \geq f(\overrightarrow{x}) & \textit{in the case of a maximization} \end{cases}$$

Figure 2.1 illustrates a global minimum (maximum) and another local minimum (maximum) for an optimization problem.

2.2 Cutting and packing problems

Cutting and packing problems (C&P) occur traditionally when we try to extract (cut) or to place (position) a predefined set of objects from (into) largest objects while respecting one or more specific constraints (for example overlapping objects, guillotine type of cutting, balancing the positioning of objects, etc.) and while optimizing a certain objective function that may depend on the containers and on the selected objects placed in the container(s) (cf. Dyckhoff *et al.* [26, 21], Wäscher *et al.* [74]).

These problems are among the best known combinatorial optimization problems. They have led to many scientific studies in the literature with a raising number of publications for many years and have many important practical applications in the industrial field (cutting of raw materials), transportation logistics (load, vehicles, trucks, pallets, etc.), storage (memory allocation problems), finance (portfolio management).

Given the importance of these problems and the high number of sub-problems related to cutting and packing, Wäscher *et al.* [74] have proposed a typology of the problems of the C&P family that takes into account the five most common problems:

1. The problem dimension: dimensions of the objects in a problem.
2. The objective: when all the available containers are not enough to contain all the objects, then we try to maximize the packing of objects in a set of available containers. Otherwise, the objective is to minimize the number of containers to be used (in the case of a single container, it consists in minimizing the dimension of the container, that can vary).
3. The objects to place: the shape, size and distribution of the objects. Three cases can be considered: (a) all the objects have the same size, (b) we only consider a small number of object classes of different sizes and, (c) the number of objects is very high and they do not have the same size.
4. The containers to be used: the nature, size and number of containers. Two cases can be examined: (a) a single container is available and its dimensions can be predetermined or vary and (b) several containers of fixed dimensions of identical sizes or different sizes.

5. The shape of the objects (items): they can be represented as circles, rectangles, cylinders, spheres, cubes, etc. Similarly, the type of the objects to pack (or cut) has the same type as the containers.

Finally, Wäscher *et al.* [74] have proposed six classes of problems of the C&P family, which are categorized according to criteria 2 and 3 previously described (cf. figure 2.2):

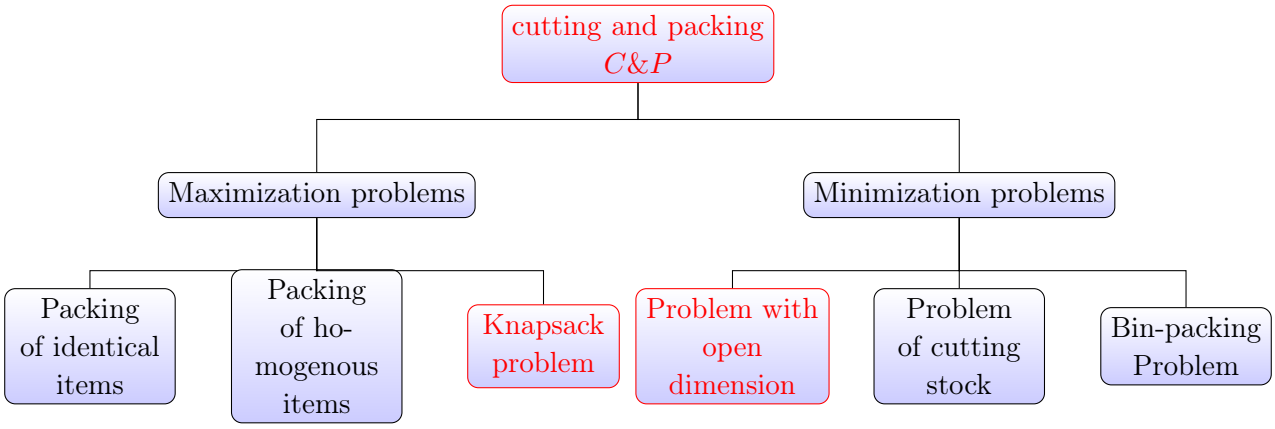


Figure 2.2: Main classes of C&P problems

In our study, we will consider the two following classes:

- Open Dimension Problems (ODP): we only use one open container or a fictional spherical container (to be determined) while the objects (items) are spherical and each of them is characterized by its radius (they can be identical or not).
- Single Knapsack Problem (SKP): the container (which can be spherical) is closed (of fixed dimensions) and the objects (items) are non-identical spheres.

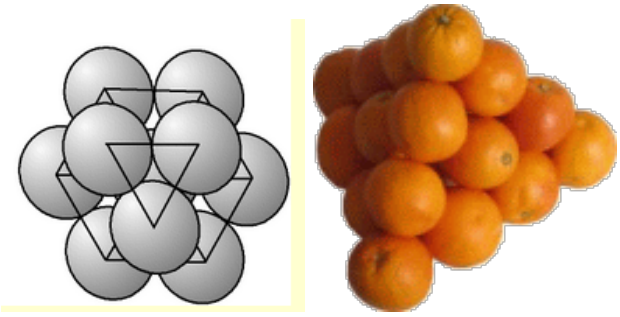
2.3 Sphere packing problem

2.3.1 Some applications

Sphere packing problems have been known since 1610, when the best known mathematical question was written (cf. Conway *et al.* [17]):

"What is the maximal density of a compact stack of identical spheres?"

In 1611, Kepler [17], a German astronomer and mathematician, formulated (without proving it) the conjecture that the maximal density was about $\frac{\pi}{3\sqrt{2}} \simeq 74.048\%$. This density is reached for a cubic packing with centered faces. This situation can be found,



(a) (b)

Figure 2.3: Structure of the packing for Kepler's conjecture

for example, in fruit and vegetable stores, as illustrated by figure 2.3.

The problem of the "Kissing Number", which consists in determining the highest possible number of identical spheres, that can be placed around a sphere of the same type, was a dispute topic between Newton (a British physicist, philosopher, astronomer and mathematician) and Gregory (mathematician and astronomer) in 1690. Newton thought that only 12 spheres could be placed around a sphere of the same size, while Gregory stated that it was 13 (cf. Conway *et al.* [17], Kucherenko *et al.* [47], Zong [79]). In 1956, Leech [49] proved that the number was 12. Hilbert (a German mathematician) included in 1900 (cf. Lagarias [48]) Kepler's conjecture in his well-known list of the 23 big unsolved mathematical problems (cf. Derbyshire [20]). In 1998, Hales wrote a proof of Kepler's conjecture with the help of computer calculations. His proof was made of a 300-pages text and 40 000 lines of code. In 2014, Kepler's conjecture was officially proved.

Practically, the applications of sphere packing are very numerous, for example:

In medicine: in the cancer treatment field, the gamma-knife (cf. figure 2.4(a)) is one of the most efficient radio-surgery techniques. The goal is to submit the cancerous region to higher doses shots (cf. figure 2.4(b)) so as to provoke reactions that can lead to the patient's recovery.

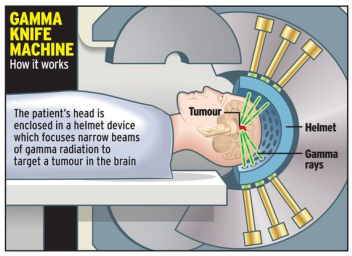
An optimal treatment by Gamma knife must verify the following criteria: (cf. Wang [73], Sutou *et al.* [69], Li *et al.* [50]):

1. The shot must not go beyond the bounderies of the region in order to avoid damaging healthy cells.

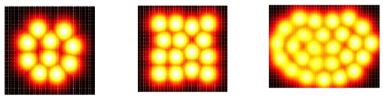
2. Overlapping between the shots are forbidden in order to avoid overdosing.

3. The best treatment is the treatment that generates the most density and avoids

(a) Gamma knife



(b) Treatment of cancers by Gamma knife



(c) Modeling of the shots and of the cancerous region

Figure 2.4: Illustrations of Gamma knife treatment

overdosing or a non-uniform distribution.

On a modeling point of view (cf. figure 2.4(c)), the shots can be considered as spheres, identical or not, and the cancerous region as a geometric shape, regular or irregular.

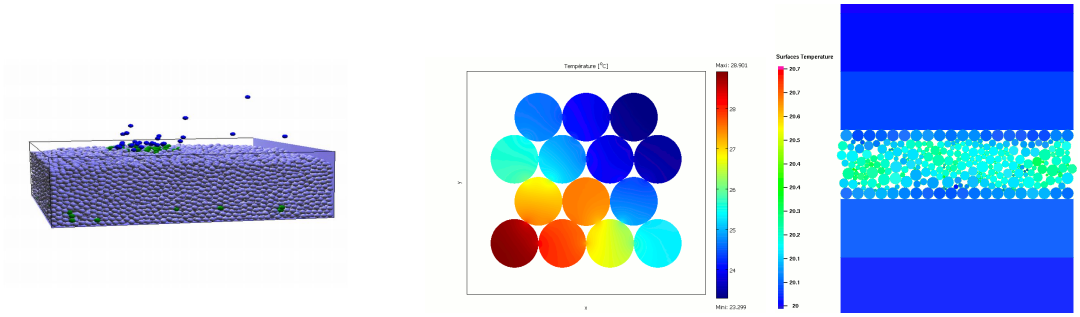
In parallel, Sutou *et al.* [69] and Li *et al.* [50], proposed the packing of non-identical spheres in a polytope to modelize the previously described treatment while Wang [73] proposed the packing of non-identical spheres in an irregular region.

Material science and biology: sphere packing has been utilised as a model to represent granular environments and the structure of lipids, proteins and vitreous materials in order to study a few properties about thermic conductivity, electrical conductivity, fluid movements, constraints distribution and other mechanical properties of granulars, living cells, etc. (cf. Williams *et al.* [75], Larrard *et al.* [18], Cheng *et al.* [12]).

In this part, we will present the sphere packing model created by Nguyen *et al.* [59, 60] in order to study the thermic conductivity in granular environments.

As shown by figure 2.5(a), granular environments are indeed generally porous and diphasic (they contain particles and pores). The particles (or grains) are modeled in the form of identical or non-identical spheres stored in a matrix (a container). The empty spaces around the particles are considered as pores, as shown by figure 2.5(b). The increase of thermic conductivity is due to a reduction of the pores' dimensions and to an increase of the space between particles.

Furthermore, Nguyen *et al.* [61] used this modeling to study the energy dissipation by friction on the interface of a braking system, which plays an important role in car and train transportation security. Their model is made of two solid layers and an intermediate



(a) Granular environment

(b) Thermic transfers by conduction between particles

(c) Fluid-particles thermic transfers

Figure 2.5: Thermic transfer in granular environments

granular layer. The latter has been modeled as sphere (or circle) packing in a container, as illustrated by figure 2.5(c).

In ecology: the modeling of the complex structure of tropical rainforests is one of the greatest challenges. One of the noteworthy questions is the distribution of different tree sizes, which is particularly important to estimate the biomass. Taubert *et al.* [70] have developed a model based on sphere packing to study and present the structure of tropical forests as illustrated by figure 2.6.

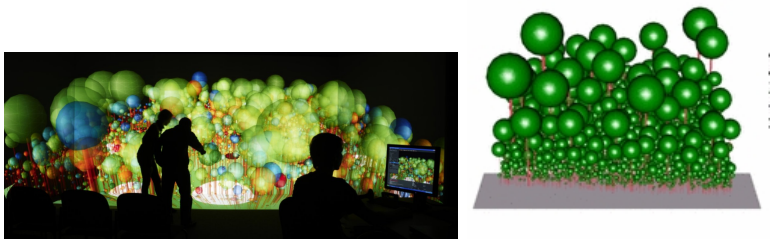


Figure 2.6: Modeling of the complex structure of tropical forests

Communication and digital storing Telephone, computer networks, translation error correction and storing error correction are considered as sphere packing problems without overlapping where we are seeking to maximize the density in a given dimension, as explained by Conway *et al.* [17] and Claudio *et al.* [13].

It should be noted that other applications are presented by Hifi *et al.* [31], where a set of problems have been handled, modeled and solved by applying various methods.

2.3.2 Some sphere packing problems

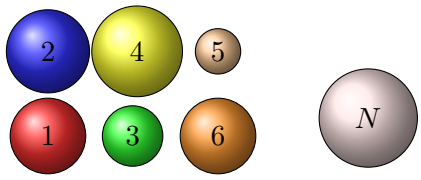
In this thesis, we are interested in solving three variants of the sphere packing problem: (i) in an open container, (ii) a closed container and (iii) a spherical container.

2.3.2.1 Packing problem in an open container

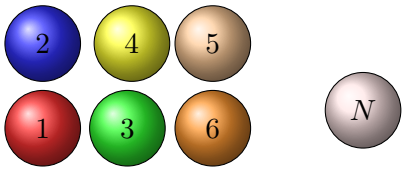
An instance of a packing problem in an open container is represented by a set N of n objects (spheres), where each object is characterized by its radius r_i (identical/non-identical) and an open container, denoted by B of dimensions (H,W,∞), where H is the width, W the depth and L= ∞ its varying length(cf. figure 2.7). The goal of the problem is to pack without overlapping all the objects (spheres) into the container so as to minimize the length L of the container.

Finding a feasible solution for this problem equals therefore to determining the values of the couple (X, L), where $X = \{x_i, y_i, z_i, ..., x_n, y_n, z_n\}$ is a vector representing the positions of the n spheres placed in the container B of length L (figure 2.8(a) illustrates a feasible solution associated with an instance of the problem).

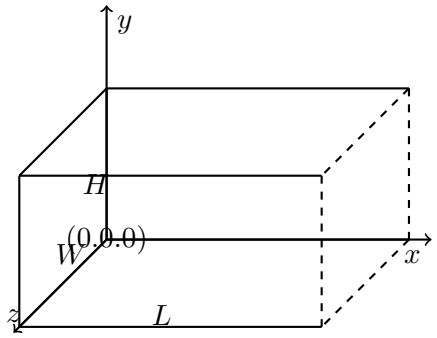
From now on, that problem will be denoted by $3D - SPP$ (Three-Dimensional Strip Packing Problem) by following the same representation as Kubach *et al.* [45, 46]).



(a) Spheres of non-identical radius



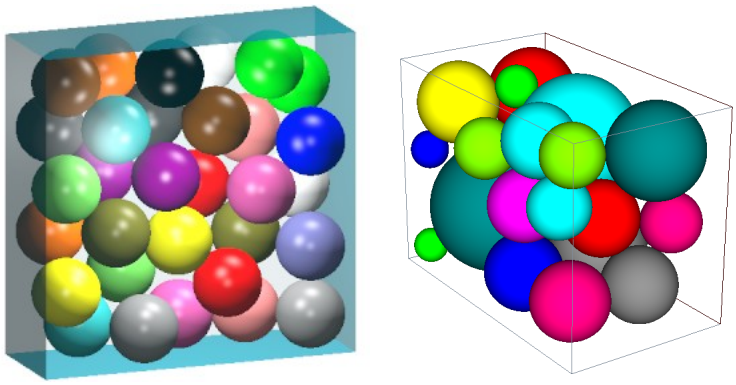
(b) Spheres of identical radius



(c) An open container B

Figure 2.7: An instance of $3D - SPP$

The problem $3D - SPP$ can be represented as the model described below (assuming that the origin is situated at point $(0, 0, 0)$):



(a) Spheres of radius $r_i = 1$ in an open container

(b) Spheres of different radius in an open container

Figure 2.8: Structures of feasible (packing) solutions in an open container B

$$\min \quad L \tag{2.4}$$

$$\delta_{(S_i,S_j)} \geq (r_{S_i} + r_{S_j}) \quad \forall (i,j) \in N^2,\ i < j \tag{2.5}$$

$$x_{S_i} \leq L - r_{S_i} \quad \forall i \in N \tag{2.6}$$

$$y_{S_i} \leq H - r_{S_i} \quad \forall i \in N \tag{2.7}$$

$$z_{S_i} \leq W - r_{S_i} \quad \forall i \in N \tag{2.8}$$

$$x_{S_i} \geq r_{S_i} \quad \forall i \in N \tag{2.9}$$

$$y_{S_i} \geq r_{S_i} \quad \forall i \in N \tag{2.10}$$

$$z_{S_i} \geq r_{S_i} \quad \forall i \in N \tag{2.11}$$

$$\underline{L} \leq L \leq \overline{L} \tag{2.12}$$

Constraints 2.5 make sure that two different spheres S_i and S_j do not overlap. Measure $\delta_{(S_i,S_j)}$ represents the Euclidean distance between two spheres.

$$\delta_{(S_i,S_j)} = \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2} \tag{2.13}$$

Constraints from 2.6 to 2.11 ensure that the spheres do not go beyond the container. $(x_{S_i}, y_{S_i}, z_{S_i})$ represents the position of the sphere i in the container B of length L. As for constraint 2.12, it represents the limitation of the optimum by a lower and an upper bounds. A lower bound $\underline{L}$ can be provided in a trivial way, for example by stating:

$$\underline{L} = \frac{4 \times \pi \sum_{i=1}^{N} r^3}{3 \times W \times H}.$$

Moreover, the upper bound $\overline{L}$ can initially either take a trivial value, such as $\overline{L} = 2 \times \underline{L}$, or be computed by a quick greedy algorithm.

Specificities of the $3D - SPP$ problem

Starting from the formulation of the $3D - SPP$ problem, we note the following points:

- The total number of variables in an instance of this problem is approximately $3n+1$.
- Most constraints are non-linear (quadratic); we have $n \times (\frac{n-1}{2})$ constraints of that type type. We also have $6 \times n$ linear constraints. The total number of constraints is approximately $6n + (N \times (\frac{n-1}{2}))$.
- The objective function (2.4), which represents the minimum length of the container to be determined, is linear. Variable L only appears in constraint (2.6). Let's suppose that we have a solution to the problem $3D - SPP$, where all the available spheres are positioned without overlaps in the container B. Then, variable L can be calculated as follows:

$$L = \max\{x_i + r_i\} \qquad \forall i \in N \tag{2.14}$$

- The penalty function, which measures the feasibility of a given solution (X, L), can be calculated from the quantities of overlapping values between two spheres or between spheres and the (fixed) dimensions of the container.

  The overlap between two spheres can be calculated as follows:

$$O_{i,j} = max\{0, r_{S_i} + r_{S_j} - \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2}\}. \tag{2.15}$$

  On the other hand, the overlaps between a sphere i and the container B on the three axes can be calculated by:

$$O_{i,X} = max\{0, r_{S_i} + |x_{S_i}| - 0.5L\} \qquad \forall i \in N \tag{2.16}$$
$$O_{i,Y} = max\{0, r_{S_i} + |y_{S_i}| - 0.5H\} \qquad \forall i \in N \tag{2.17}$$
$$O_{i,Z} = max\{0, r_{S_i} + |y_{S_i}| - 0.5W\} \qquad \forall i \in N \tag{2.18}$$

  Finally, the penalty function $E(X, L)$ is calculated by adding the characteristic values of all overlaps, as follows:

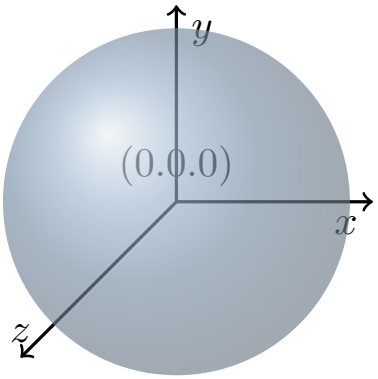
$$E(X, L) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} O_{i,j}^2 + \sum_{i=1}^{N} (O_{i,x} + O_{i,y} + O_{i,z}) \tag{2.19}$$

  A solution (X, L) is considered as a feasible solution *if and only if* $E(X, L) = 0$.

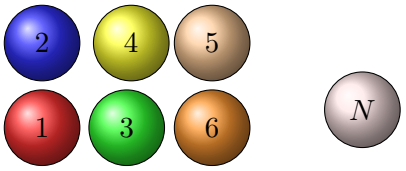
2.3.2.2 Sphere packing problem: using a spherical open container

An instance of the packing problem in a spherical container is represented by a set N of n objects (spheres), where each object is characterized by its radius ($r = 1$), and a spherical container of unlimited radius ($R = \infty$) (cf. figure 2.9). The goal of the problem is to place without overlaps all the (spherical) objects in the container so that the radius R of the container is minimized.

This problem, denoted by $3D - PSS$ (Three-Dimensional Packing Smallest Sphere) has been studied by M'Hallah *et al.* [54]). Figure 2.10 illustrates a feasible solution of problem $3D - PSS$.



(a) A spherical container S



(b) A set of spheres of identical radius

Figure 2.9: An instance of $3D - PSS$ problem

Figure 2.10: Structures of an admissible solution for the placement in a sphere

Formally, the $3D - PSS$ problem can be represented as follows (cf. M'Hallah *et*

al. [54]) :

$$\min \quad R \tag{2.20}$$

$$\delta_{(S_i,S_j)} \geq 2 \quad \forall (i,j) \in N^2,\ i < j \tag{2.21}$$

$$\delta_{(S_i,S)} \leq R - 1 \quad \forall i \in N \tag{2.22}$$

Constraints 2.21 check that two spheres do not overlap; in this case, the Euclidean distance $\delta_{(S_i,S_j)}$ between the centres of two spheres must be higher than the sum of their radius. As for constraint 2.22, it ensures that sphere i is entirely contained inside the spherical container (also called the fictional sphere), where $\delta_{(S_i,S)}$ represents the Euclidean distance between the centre of the object (sphere) i and the centre of the spherical container. We will distinguish the two following cases:

(i) The origin is situated in the middle of the spherical container R. Then:

$$\delta_{(S_i,S)} = \sqrt{x_{S_i}^2 + y_{S_i}^2 + z_{S_i}^2} \tag{2.23}$$

(ii) The origin is not situated in the middle of the spherical container R. Then:

$$\delta_{(S_i,S)} = \sqrt{(x_{S_i} - x_S)^2 + (y_{S_i} - y_S)^2 + (z_{S_i} - z_S)^2} \tag{2.24}$$

Specificities of $3D - PSS$ problem

- The total number of variables in an instance of this problem is approximately $3n+1$.
- Most constraints are non-linear (quadratic); we have $n + n \times (\frac{n-1}{2})$ constraints of this type.
- The objective function 2.20 represents the radius of the spherical container S to be optimized (that we are trying to minimize). We find that this radius only appears in constraint 2.22. Now, let us assume that we have a solution (X, R) of 3D-PSS, where all available spheres are positioned without overlapping in S. Then, the radius R is calculated as follows:

$$R = max\{\sqrt{x_{s_i}^2 + y_{s_i}^2 + z_{s_i}^2} + 1\} \quad \forall i \in N \tag{2.25}$$

- The penalty function $E(X, R)$ for a given solution (X, R) can be calculated from the overlaps as followed:
  - the overlap between two spheres is:

$$O_{i,j} = max\{0, 2 - \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2}\} \tag{2.26}$$

– the overlap between a sphere and the spherical container is:

$$O_{i,S} = max\{0, R - \sqrt{x_{S_i}^2 + y_{S_i}^2 + z_{S_i}^2} + 1\} \tag{2.27}$$

– The penalty function $E(X, R)$ is finally:

$$E(X, R) = \sum_{i=1}^{N} \sum_{j=0}^{N} O_{i,j}^2 \tag{2.28}$$

Note that a solution (X, R) is said to be feasible **if and only if** $E(X, R) = 0$.

2.3.2.3 Sphere packing problem: using a closed container

A packing instance in a closed container is represented by a set N of n objects (spheres), where each sphere is characterized by its radius r_i, and a closed container P represented by its dimensions (L, W, H). Often, the container P is not big enough to contain all the N available spheres. Hence, the aim is to maximize the density D_{max} by placing without overlaps a subset $\overline{N}$ of N. Note that if $\overline{N} = N$, then we have an optimal solution for the instance.

Also note that, in a more general version, each of the spheres can be represented by a weight (or a score /profit). The goal, in this case, is to maximize the weight of the spheres that we want to pack. A feasible solution to the problem is represented by figure **??**.

This problem can be considered as a variant of the Three-Dimensional Knapsack Problem $(3D - KP)$ as it has been studied in Kubach *et al.* [45, 46]. It can also be modeled as shown below (the model below considers that the reference is situated at the point (Bottom, Behind, Left) of coordinates $(0, 0, 0)$):

$$\max \quad \alpha \sum_{i \in N} r_i^3 \xi_i \tag{2.29}$$

$$\text{s.t.} \quad \xi_i r_i \leq x_i \leq \xi_i (L - r_i), \ i \in N \tag{2.30}$$

$$\xi_i r_i \leq y_i \leq \xi_i (H - r_i), \ i \in N \tag{2.31}$$

$$\xi_i r_i \leq z_i \leq \xi_i (W - r_i), \ i \in N \tag{2.32}$$

$$\xi_i \xi_j (r_i + r_j) \leq \sqrt{(x_i - x_j) + (y_i - y_j) + (z_i - z_j)}, \ (i, j) \in N^2, i < j \tag{2.33}$$

$$\xi_i \in \{0, 1\}, \ i \in N \tag{2.34}$$

$$\tag{2.35}$$

where $\alpha = (4\pi)/(3LHW)$.

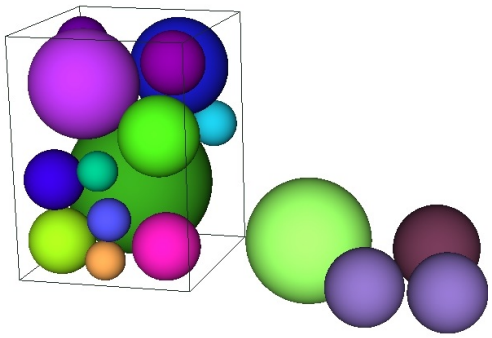


Figure 2.11: Structures of feasible solutions-Closed Container

Specificities of 3D-KP

Using the same method as in the previous sections (for the other two variants of the problem), we can analyse the spatial complexity of the given model.

- The total number of variables in a given instance of the $3D-KP$ problem is $3n+1$. Most constraints are non-linear (quadratic); we have $n\times(\frac{n-1}{2})$ non-linear constraints. We also have $6 \times n$ linear constraints. The total number of constraints is approximately $6n + (N \times (\frac{n-1}{2}))$.
- Each constraint is associated with a decision variable ξ_i such that:

$$\xi_i = \begin{cases} 1 & \text{sphere } i \text{ is placed in the container } P \\ 0 & \text{otherwise} \end{cases}$$

- The objective function (2.29) represents the density D of a feasible solution. The latter is defined as the proportion of the volume of P occupied by the spheres inside where

$$\alpha = (4\pi)/(3LHW) \tag{2.36}$$

- The penalty function $E(X, P)$ and the overlaps are calculated as in the case of the packing in a strip (equations of 2.15 to 2.19).

Remarks

- The variants of the sphere packing problem are NP-hard and the number of local minima is far superior to $N!$ (cf. Stoyan *et al.* [67], Kubach *et al.* [46]). Therefore, there is no algorithm that can solve the problem optimally within a polynomial time. Of course, our goal is not to study thoroughly the algorithmic complexity of these

problems since we are mainly interested with heuristics based upon experimental methods.

- The classification of this "combinatorial or continuous" problem is fuzzy (cf. Hifi *et al.* [31]). Indeed, it belongs to both combinatorial optimization problems and continuous optimization problems, because the structure of a feasible solution is of a discrete nature while the positions of the spherical objects to be placed in a container are of continuous nature.

  In that case, methods can be applied by keeping only the combinatorial aspect even if the positioning of the objects remains continuous. In other cases, continuous optimization methods are applied and the combinatorial aspect can be neglected since, for example, descent methods primarily focus on the continuous aspect of variables (cf. Hifi *et al.* [31]).

2.4 Conclusion

In this chapter, we described the general frame of the problems handled in this manuscript. We described a classification used in the literature for cutting and packing problems and, the position of the variants studied in this work. We also discussed different variants of the sphere packing problem and their applications, e.g., sphere packing in an open container, in a closed container and in a spherical container.

Chapter 3

Solution methods for packing problems

Contents

This chapter presents some of the well-known solution methods developed for solving some sphere/circle packing problems. Because the sphere/circle packing problem belongs to the NP-hard class, the approaches used for tackling such problems in the literature are often based on experimental methods (specific heuristics and meta-heuristics), methods using simulations and hybrid methods. Herein, we first describe other variants of the packing problem. Second and last, we try to describe some proposed approaches available in the literature that are used for approximately solving several variants of the packing problem.

3.1 Introduction

Since the sphere/circle packing problem belongs to the NP-hard class, no algorithm can solve it optimally in a polynomial time. Consequently, approaches used in the literature are often based on experimental methods. These methods include specific heuristics, meta-heuristics, methods using simulation and hybrid methods. Hifi and M'Hallah [31] proposed an interesting review on two- and three-dimensional circular/spherical packing problems, where Euclidean distance is used.

Before describing some well-known methods used for solving the sphere packing and its related problems, we first propose complementary problems which are encountered in several real-world applications and recently tackled in the literature. These problems may be viewed as variants of the sphere packing described in section 3.2: packing spheres into a cuboid, into a cube, into a cylinder, into a pyramid, and into a regular tetrahedron (cf. Hifi *et al.* [31], Birgin and Sobral [8]).

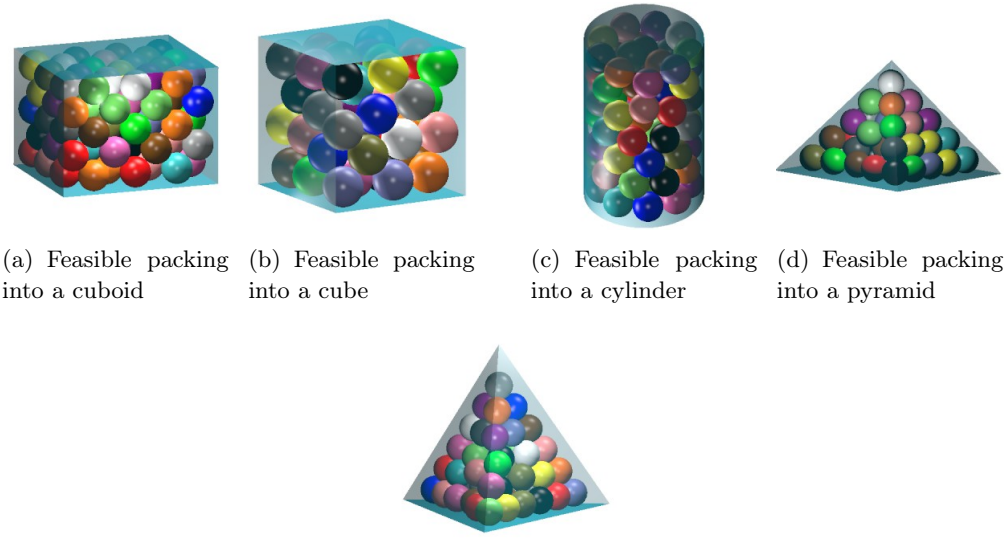
Variant	Model	Nb variables	Nb constraints
Cuboid	$\min L.W.H$ $S.t.\ r_i \leq x_i \leq L - r_i, \forall i \in N$ $r_i \leq y_i \leq H - r_i, \forall i \in N$ $r_i \leq z_i \leq W - r_i, \forall i \in N$ $\delta_{(i,j)} \geq (r_i + rj)$	$3N + 3$	$6n + (N \times (\frac{n-1}{2}))$
Cube	$\min W$ $S.t.\ r_i \leq x_i \leq W - r_i, \forall i \in N$ $r_i \leq y_i \leq W - r_i, \forall i \in N$ $r_i \leq z_i \leq W - r_i, \forall i \in N$ $\delta_{(i,j)} \geq (r_i + rj)$	$3n + 1$	$6n + (N \times (\frac{n-1}{2}))$
Cylinder	$\min R^2 H$ $S.t.\ x_i^2 + y_i^2 \leq (R - r_i)^2, \forall i \in N$ $R \geq r_{max} \equiv max\{r_i\}, \forall i \in N$ $r_i \leq z_i \leq H - r_i, \forall i \in N$ $\delta_{(i,j)} \geq (r_i + rj)$	$3n + 2$	$3n + 2 + (N \times (\frac{n-1}{2}))$
Pyramid	$\min L$ $S.t.\ 2x_i + \sqrt{2}z_i \leq L - \sqrt{6}r_i, \forall i \in N$ $-2x_i + \sqrt{2}z_i \leq L - \sqrt{6}r_i, \forall i \in N$ $2y_i + \sqrt{2}z_i \leq L - \sqrt{6}r_i, \forall i \in N$ $-2y_i + \sqrt{2}z_i \leq L - \sqrt{6}r_i, \forall i \in N$ $z_i \geq 0, \forall i \in N$ $\delta_{(i,j)} \geq (r_i + rj)$	$3n + 1$	$5n + 1 + (N \times (\frac{n-1}{2}))$
Tetrahedron	$\min L$ $S.t.\ 2\sqrt{2}x_i - 2\sqrt{6}y_i + 2z_i \leq \sqrt{6}L - 6r_i, \forall i \in N$ $2\sqrt{2}x_i + 2\sqrt{6}y_i + 2z_i \leq \sqrt{6}L - 6r_i, \forall i \in N$ $-2\sqrt{2}x_i + z_i + r_i \leq 0, \forall i \in N$ $z_i \geq 0, \forall i \in N$ $\delta_{(i,j)} \geq (r_i + rj)$	$3n + 1$	$4n + 1 + (N \times (\frac{n-1}{2}))$

Table 3.1: Illustration of the formal models when varying the container.

3.2 Complementary packing problems

In this section, we present other variants of the sphere packing family. Depending on the container used to pack a set of available (identical or non-identical) spheres/items, we can distinguish the problems related to packing spheres/items into a specified container, such as a cuboid, a cube, a cylinder, a pyramid, and a tetrahedron.

Table 3.1 illustrates the problems characterized by Birgin and Sobral [8], where each version of the problem is represented by its model. Column 1 refers to the version of the problem, column 2 reports the formal description of that problem, and columns 3 and 4 tally some specificities related to the number of variables and constraints associated with the instance of the problem.



(a) Feasible packing into a cuboid

(b) Feasible packing into a cube

(c) Feasible packing into a cylinder

(d) Feasible packing into a pyramid

(e) Feasible packing into a tetrahedron

Figure 3.1: Sphere packing into different shapes

From table 3.1, one can observe that:

1 - $\delta_{(i,j)} = \sqrt{(x_{S_i} - x_{S_j})^2 + (y_{S_i} - y_{S_j})^2 + (z_{S_i} - z_{S_j})^2}$ represents the Euclidean distance between two spheres. It is a non-linear (quadratic) constraint and there are $n \times (\frac{n-1}{2})$ constraints of that type.

2 - The other (linear) constraints ensure that the spheres do not go beyond the container; (x_i, y_i, z_i) represents the position of the sphere i in the container.

3 - The objective function of each model is to minimize the dimension of the container. Note that all of these variants belong to the class of "open dimension problems".

4 - The cartesian system is considered for each of these models.

5 - Figure 3.1 (extracted from Birgin and Sobral [8] and available on the following website:
https://www.ime.usp.br/∼egbirgin/packing)) illustrates the feasible solutions for the considered variants.

Because the sphere packing problem is simply an extended version of the circle packing problem, each of these models (cf. table 3.1) can be reformulated when using a polar coordinate system and (or) a mixed polar/cartesian system (for more details, the reader can be referred to Mladenović *et al.* [57], Lopez [52], and Hifi *et al.* [31].

3.3 Some solution methods

The sphere/circle packing problem belongs to the NP-hard combinatorial optimization problems, for which most approaches are based on approximate methods (heuristics, meta-heuristics, simulation-based methods and hybrid methods). Generally, approximate methods are an alternative to exact methods for tackling complex optimization problems where large-scale or complex instances are considered. The goal of such methods is to achieve solutions with good quality (not necessarily optimal) within a reasonable time. As underlined above, they become interesting, especially for solving large-scale instances, whenever the runtime may grow with the size of the instance or according to its complexity.

We note that the effectiveness of these type of methods may be analyzed depending on both its achieved solution's quality and its runtime, which may be compared to the best solutions available in the literature when using an equivalent runtime limit.

3.3.1 Heuristics

The advantage of a heuristic is related to its simplicity to construct a series of solutions within a reasonable (small) runtime. Constructive heuristics are often used in the literature in order to find moderate solutions within small average runtime. Indeed, George *et al.* [23] have developed several rules to create constructive heuristics allowing to measure at the end the quality of the achieved solutions. Among these rules, we can find the sorting strategy that orders all items in decreasing order of their radii; its goal is to favor items with the largest radius, while the others (with the smallest radii) are used for filling the unused spaces. Such a constructive heuristic consists in filling a container by positioning items step by step until no item, from the remaining items to be packed, can be added to the current solution. Generally, the item to be positioned into the container is given by a function, like a local choice criterion. Then, the best item realizing the favored position

is packed into the container. Such a process tries to provide a partial solution that is iteratively completed by packing an item belonging to the non-packed items.

Of course, several criteria related to a local choice have been suggested in the literature. Indeed, most of these criteria are mainly based upon searching the position realizing the *minimum local distance position* between some items and the container (as described in chapter 4, section 4.2.2). For the sphere/circle packing problem, two constructive heuristics can be distinguished: (i) constructive greedy procedures and (ii) constructive procedures using a look-ahead (beam search).

3.3.1.1 Constructive greedy procedures

The Constructive Greedy Procedure (noted CGP) starts by positioning the first item into the container, while the $n-1$ remaining items are successively positioned according to a local choice criterion. In the case of different eligible positions (positions realizing the same distance), then the chosen item to pack is either (i) the first item realizing the smallest index related to its order or (ii) the item realizing the smallest distance from the edge the container. In what follows, the framework of the basic greedy procedure used for building a starting feasible solution is described.

1. Let $I_i = \{i\}$ be the set of items of N already packed in the current container. Set $\overline{I_i} = \{1, .., n\} \setminus \{I_i\}$ as the set of items of N which are not yet packed in the container and let P_{I_i} be the set of eligible positions of the i-th item.

2. Update P_{I_i} with the eligible positions of each $j \in \overline{I_i}$.

3. Repeat

   (a) Let p_k^{i+1} be the best position of the next item to pack according to the local choice criterion.

   (b) Position the $(i+1)^{th}$ item into the position p_k^{i+1} and move the packed item to I_i.

   (c) Update the set of eligible positions P_{I_i}.

   Until $P_{I_i} = \emptyset$

4. Exit with a feasible solution and the subset $I_i \subseteq N$ containing the packed items.

Huang *et al.* [39] have developed two greedy constructive heuristics (noted $B1.0$ and $B1.5$) for packing non-identical circles into a rectangular container. These heuristics use a local choice criterion called *Maximum Hole Degree* (noted MHD) which allows to evaluate the best position of an item by evaluating the unused area when positioned at the corner of the corner position of the container. The heuristic $B1.0$ has been used in order to build a feasible solution by iteratively positioning the circles into the container while the

heuristic $B1.5$ improves the solutions by using a look-a-head strategy; that is used for searching a series of feasible solutions following a series of paths, where the selected item to pack is that realizing the best solution over all developed paths.

Kubach *et al.* [46] used an adaptation of $B1.0$ and $B1.5$ for solving the sphere packing problem and by considering both versions of the problem: open and closed container. Because of the huge number of eligible positions of each candidate item to pack, parallel versions of Kubach *et al.*'s [46] algorithms have been proposed (cf. Kubach *et al.* [45]).

Akeb ([3, 2]) proposed a simple adaptation of the look-a-head search (Huang *et al.* [39]) with the MHD for tackling the non-identical sphere packing problem.

Hifi *et al.* [27] proposed two heuristics for the non-identical circle cutting problem in a rectangular container. The first heuristic is a greedy one that uses the adapted best local position which tries to reach a better set of eligible positions. Such a criterion can be viewed as a variant of the bottom-left strategy since it chooses the left-most and highest position to next item to pack. The second heuristic combines the genetic algorithm with a constructive procedure: the genetic algorithm is used to create a series of new order of items to pack, whereas the constructive procedure was used to reach a feasible solution for the problem. Both procedures were iterated until providing a satisfactory solution for the problem.

3.3.1.2 Constructive procedure using a look-ahead (beam search)

The Beam Search (noted BS) has been first proposed by Ow and Morton [62] for tackling the scheduling problem and it has since been successfully applied to many other combinatorial optimization problems (Akeb and Hifi [4, 6, 5, 7]). Such an approach can be viewed as a truncated tree search procedure where the objective is to avoid an exhaustive search by performing a partial enumeration of the solution space.

At each level of the developed tree, only a subset of nodes (called the *set of elite nodes*) are selected for further branching, while the other nodes are discarded; no backtracking is performed. For each level, the cardinality of the elite nodes to be investigated is set to ω, which is called the *beam width*. Generally, these selected ω nodes represent those having a high potential to lead the best solutions for the treated problem. Furthermore, each node is assessed via an *evaluation function* whose role is to provide a promising separation mechanism of the nodes of each level of the developed tree.

Algorithm 1 describes the main steps of the standard beam search which is characterized by a beam width ω used for filtering the set of offspring nodes B_ω. A node corresponds to a partial feasible solution and the set B of current nodes is initialized to the root node B_0, whereas B_ω, containing the offspring nodes, is initialized to the empty set. Each node of B generates a set of offspring nodes, and moves them to B_ω. If a node η of B_ω is a leaf (i.e. no further branching is possible out of η), then its objective function value z_η is computed and compared to $z^\star$. If $z_\eta > z^\star$, then the incumbent solution is set to the

Algorithm 1 Beam Search

1. **Initialization step**
   (a) Let ω be the beam width.
   (b) Set $B = \{B_0\}$ and $B_\omega = \emptyset$, where B is the set of nodes to be investigated, and B_ω the set of nodes branched out of the nodes in B.
   (c) If an initial feasible solution is available, set $z^\star$ to its objective function value; otherwise, set $z^\star = +\infty$ (minimization problem).
2. **Iterative step**
   Repeat
   (a) Choose a node $\eta \in B$; branch out η; remove η from B and insert the created nodes into B_ω.
   (b) If a node η of B_ω is a leaf, then
      i. compute its objective function value z_η;
      ii. if $z_\eta > z^\star$, update $z^\star$ and the incumbent solution;
      iii. remove η from B_ω.
   (c) Assess the potential of each node of B_ω using an evaluation function.
   (d) Rank the nodes of B_ω in a decreasing order of their values.
   (e) Insert the $\min\{\omega, |B_\omega|\}$ best nodes (called elite nodes) of B_ω into B; and set $B_\omega = \emptyset$.
   Until $B = \emptyset$.

leaf node; $z^\star$ is then updated: $z^\star = z_\eta$; and η is removed from B_ω. All nodes belonging to B_ω are assessed using an evaluation function, and then ranked in decreasing order of their values. The first ω nodes of B_ω are then considered as the elite nodes which are appended to the set B, whereas the remaining nodes of B_ω are discarded and B_ω is reset to the empty set. This process is iterated until no further branching is possible, i.e. until $B = \emptyset$.

3.3.2 Meta-heuristics

Meta-heuristics are mainly designed for approximately solving a wide range of hard (combinatorial) optimization problems (cf. Boussaïdare *et al.* [11]). A metaheuristic can be viewed as a generic (general) algorithm which can be adapted to different optimization problems. Its main feature is to achieve a balance between intensification and diversification strategies to converge toward satisfactory solutions. Diversification consists in generating diverse solutions to explore in the search space, while intensification focuses on a series of local regions where good solutions have been located.

There are several meta-heuristics whose differences are generally related to the representation of an instance of the problem to solve, the repartition of the starting solutions (in the population), the path to reach new solutions, how the solutions replace the old ones, the stopping criteria, etc.

Some of these meta-heuristics may have some common characteristics (cf. Boussaïdare *et al.* [11], Blum *et al.* [10],Blum *et al.* [9]), like:

- Nature-inspired (physics, biology or sociology).
- Stochastic approaches (involving random variables).
- No use of the gradient or Hessian matrix of the objective function.
- Parameters setting to fit to the problem at hand.

According to the latest survey proposed by Boussaïdare *et al.* [11], the meta-heuristics may be classified into two distinct categories:

1. Trajectory methods which start with a single initial solution and move away from it.
2. Population-based meta-heuristics which manipulate a population of solutions at each iteration.

In the following, we describe some of the meta-heuristics used to solve the circle/sphere packing problem.

3.3.2.1 Trajectory methods

These approaches are also called single-solution based approach (cf. Boussaïdare *et al.* [11]). They start with a single initial solution and move away from it. Among the solution approaches belonging to this category, we can find the simulated annealing method, the tabu search, the variable neighborhood search, the iterated local search, etc.

A- Simulated annealing:

Simulated annealing (noted SA) models the physical process of heating a material and then slowly lowering the temperature until it achieves a solid state of minimal energy E. This process has been introduced as a resolution technique of combinatorial optimization problems by Kirkpatrick *et al.* [44]. The main steps of such technique are described in algorithm 2.

Algorithm 2 Simulated annealing (SA)

1: **Initialization step**
2: Current (initial) solution S
3: Initialize the temperature T
4: Iterative step
5: **while** $\Big($Stopping criterion is not satisfied$\Big)$ **do**
6: **repeat**
7: Randomly select $S' \in N(s)$
8: **if** $f(s') \leq f(s)$ **then**
9: $S \leftarrow S'$
10: **else if** $f(s') > f(s)$ **then**
11: $S \leftarrow S'$ with probability $P = e^{(\frac{-f(s')-f(s)}{T})}$
12: **end if**
13: **until** Thermodynamic equilibrium of the system is reached
14: Decrease T
15: **end while**
16: Return the best solution found

First, SA starts with an initial solution S, and a controllable parameter T (temperature). Then, at each iteration of SA, several transformations are applied to the current solution S in order to generate a neighbor solution $S' \in N(s)$ (line 7). Consequently, the new generated solution S' is accepted to replace the current one S according to its objective function (line 6) or according to the acceptance probability $P = e^{(\frac{-f(s')-f(s)}{T})}$ (line 10). Finally, the temperature T is decreased (line 14) until a thermodynamic equilibrium of the system is reached. Note that, the acceptance probability P allows to explore a greater part of the search space and to avoid to getting trapped too quickly in a local optimum.

Zhang *et al.* [77] have combined simulated annealing with Tabu Search (noted TS) to pack non-identical circles (resp. identical) in a circular container. SA was introduced to escape from local optima with probability mechanism. TS was mainly used for preventing cycling and enhancing diversification. The initial solution S has been (generally) randomly generated. The neighborhoods were made by selecting a circle with the greatest deformation and randomly move it inside the container, then a continuous local optimization was used to improve the quality of the new generated solutions $S' \in N(s)$.

Hifi *et al.* [35] have presented a SA to solve the (non)identical circles cutting problem on a rectangular strip. The approach is based upon an energy function that, when it becomes minimum, provides solutions in which a set of items concentrated at the bottom-left corner of the original strip. The neighborhoods are generated by moving circles (horizontally, vertically and diagonally) and by exchanging some circles.

B- Tabu Search – TS

TS is based on the idea 'learn from the past' by using of mechanisms inspired by the human memory (history of the search). The method was presented firstly by Glover and Laguna [24]. The main idea is to move from a solution S toward another neighbor solution S' where the return to already explored solutions is forbidden for a certain number of iterations by keeping them into a temporary list (called tabu list).

Algorithm 3 Tabu search

Input Current (initial) solution S

Output Best solution encountered so far.

1: **Initialization step**
2: $Tabulist \leftarrow \phi$
3: Iterative step
4: **while** (Stopping criterion is not satisfied) **do**
5: select best $S' \in N(s)$
6: **if** $f(s') \leq f(s)$ **then**
7: $S \leftarrow S'$
8: Update $Tabulist$
9: **end if**
10: **end while**
11: Return the best solution found

Algorithm 3 describes the main steps of the standard tabu search. At each iteration of the algorithm, the best neighbor solution $S' \in N(s)$ (line 5) is selected as a new solution and kept in a tabu list for a certain number of iterations (line 8). The list can be viewed as short-term memory that records information on recently visited solutions. The structure of this list is considered as a critical point. For example,

- The length of the tabu list controls the memory of the search process. If this length is low, the search will concentrate on small areas of the search space (intensification), a high length forces the search process to explore larger regions (diversification) whereas a dynamic variation of the length may lead to a more robust tabu search algorithm which balances between the exploration and exploitation mechanisms.

- On the other hand, the use of this list can prevent attractive moves. Therefore, the aspiration criteria must be used as an improvement strategy, where a move which leads to a better solution does not have any reason to be prohibited.

For the circle/sphere packing problem, Fu *et al.* [22] have proposed an iterated tabu search approach (ITS) to solve the circle packing problem in a strip. ITS starts from a randomly generated solution and attempts to gain improvements by a tabu search procedure. During the search, if the obtained solution is not feasible, a perturbation operator is subsequently employed to reconstruct the incumbent solution. The authors

have used several perturbation strategies: (i) dividing the circles into two subgroups (big circles and small circles) and exchanging the positions of two circles belonging to the same group. (ii) Dropping the current solution by removing the small circles out the container and reinserts them randomly. Therefore, a tabu list is introduced to forbid the previously swapped circles to be re-swapped within a certain number of iterations. The tabu list was presented by an integer array TabuTenure[N] (N is the number of available circles) which records the tabu tenures of all the circles. At each iteration of the TS procedure, if the best neighboring solution corresponds to swapping circles i and j, then the tabu tenures of both circles was updated respectively as follows:

$$TabuTenure[i] = CurIterNum + T + rand(0, \frac{N}{8}) \tag{3.1}$$

$$TabuTenure[j] = CurIterNum + T + rand(0, \frac{N}{8}) \tag{3.2}$$

Where T is a constant which is experimentally fixed to the value 2. An aspiration criterion is further employed to override the forbidden rule: if a swapping leads to a solution better than (with lower penalty function) the best solution found so far; the generated neighboring solution will be admitted as a candidate solution, no matter the swapped circles are tabu or not. Finally, a limited-memory LBFGS algorithm was used as a continuous local optimization to build a feasible solution.

Zeng *et al.* [78] have proposed a hybrid approach between iterative tabu search and variable neighborhood descent for packing unequal circles into a circular container. The proposed tabu search only uses the swap operator for generating a neighborhood. The tabu tenures of both swapped circles i and j was updated respectively as follows:

$$TabuTenure[i] = \frac{N}{5} + rand(0, 10) \tag{3.3}$$

$$TabuTenure[j] = \frac{N}{5} + rand(0, 10) \tag{3.4}$$

The limited-memory LBFGS algorithm is also used to re-built a feasible solution.

C- Variable neighborhood search – VNS

Variable neighborhood search (noted VNS) is a meta-heuristic or a framework for building heuristics. It has been proposed by Hansen and Mladenovic ([56], [**?**]), whose main idea relies on a systematic change of the neighborhood structure for a given solution. This idea is inspired fro the three following observations: (i) a local minimum relatively to one neighborhood structure is not necessarily so for another neighborhood, (ii) a global minimum is a local minimum relatively to all possible neighborhood structures and (iii) a local minima to one or several neighborhoods are relatively close to each other.

Algorithm 4 Variable neighbourhood search VNS

1: **Initialization step**
2: Select a set of neighborhood structures $N_k, k = 1, 2, ...k_{max}$
3: An initial solution S
4: Iterative step
5: **while** (Stopping criterion is not satisfied) **do**
6: k=1
7: **while** $\left(k < k_{max}\right)$ **do**
8: shaking step: select randomly S' in the n^{th} neighborhood of S
9: local search step: come up with S''
10: **if** S'' better than S **then**
11: $S'' \leftarrow S'$
12: k=1
13: **else**
14: k=k+1
15: **end if**
16: **end while**
17: **end while**
18: Return the best solution found

Algorithm 4 describes the main steps of the standard VNS. At the initialization step, an initial solution S is generated and a set $N_k, k = 1, 2, ...k_{max}$ of neighborhood structures is defined. The main loop of VNS (line 5) controls the search time of VNS algorithm. The second loop (line 7) controls the set of neighborhood structures used ($k = 1...k_{max}$). Then, three iteratively steps are used: shaking step (line 8), local search step (line 9) and move step (line 10). In the shaking step, a solution S' is randomly selected in the n^{th} neighborhood of the current solution S. Then S' is considered as an input solution for local search step. At the end of the local search, a new solution S'' is generated. If S'' is better than S, then S'' replaces S and the loop starts again with first neighborhood structures ($k = 1$). Otherwise, the algorithm moves to the next neighborhood structure $k + 1$.

M'Hallah *et al.* [54, 55] have proposed VNS to solve the identical sphere into a smallest sphere and cube. In these works, the initial solutions are: (i) feasible solutions (generated constructively or randomly) and (ii) non-feasible solutions (generated randomly). The set of neighborhood structures was defined by changing the position of one sphere (or K spheres) closest to the center of the container. A Non-Linear Programming with Non-Monotone and Distributed Line Search (NLPQLP) solver was used as a local search to find a local optimum S'' (recent surveys of VNS and its variants is available in Hansen and Mladenovic [**?**] Mladenovic *et al.* [**?**]).

D- Iterated local search – ILS

Iterated local search (noted ILS) is based on two principal mechanisms: (i) a perturbation procedure which modifies the current solution to get a new one not too different from the initial solution, and (ii) a local search which improves the perturbed solution until reaching a local optimum.

Algorithm 5 Iterated local search (ILS)

1: **Initialization step**
2: An initial solution S
3: Apply local search starting from S to come up with S'
4: **repeat**
5: Perturb S' to get S''
6: Apply a local search starting from S'' to come up with S^*
7: **if** the acceptance criterion is satisfied **then**
8: $S' \leftarrow S^*$
9: **end if**
10: **until** stopping criterion is not satisfied
11: Return the best solution found

The main steps of the standard iterated local search are displayed in algorithm 5. It starts with an initial feasible or non-feasible solution S (line 2). First, the local search is called to get a local optimum S' (line 3). Then, at each iteration of the ILS, a new perturbed solution S'' is generated (line 5) and it is improved by calling the local search (line 6). Finally, the new solution produced by local search S^* is replaced with S' if it is better (line 8).

Huang *et al.* [40] proposed ITS for packing unequal circles into a circular container by using Critical Element Guided Perturbation (noted CEGP) strategy to jump out the local minima. CEGP is based on moving a circle, having more quantity of overlap, from its position to another position and using limited-memory based method (as a local search procedure) to get a local optimum.

Other algorithms belonging to trajectory methods family, such as the guided local search, the greedy randomized adaptive search procedure, are explained in detail in latest survey on optimization meta-heuristics proposed by Boussaïdare *et al.* [11].

3.3.2.2 Population-based approaches

Population-based approaches are divided into two categories depending on their nature-inspired (biology or sociology):

- Evolutionary Computation (EC), where the population of individuals is modified through crossover and mutation operators. Among the algorithms in this category, we can cite: Genetic algorithms, evolution strategy, evolutionary programming, co-evolutionary algorithms, differential evolution, scatter search, path relinking, etc.

- Swarm Intelligence (noted SI) which is based on the following observation: local interactions between individuals often lead to the emergence of global and self-organized behavior. This observation has been lead to produce computational intelligence by exploiting simple analogs of social interaction. Among the algorithms belong to this category, we can cite: particle swarm optimization, ant colony optimization, bacterial foraging optimization, bee colony optimization, artificial immune systems, etc.

Among those used in circle/sphere packing, there are genetic algorithm which belong to the first category, and particle swarm optimization, which belong to the second one. Consequently, in the following, we will show only these algorithms, the other algorithms are explained in detail in the latest survey on optimization meta-heuristics proposed by Boussaïdare *et al.* [11].

A- Genetic algorithms

Genetic algorithms (noted GA) is the most well-known and mostly used evolutionary computation family EC. It is a stochastic method, based on the Darwin theory of evolution, which at each generation (iteration), it selects parents and uses them to produce the children (by using a crossover and a mutation operators) for the next generation.

Algorithm 6 Standard Genetic Algorithm (GA)

1: **Initialization step**
2: Initialial population with random solutions (chromosomes)
3: Evaluate each chromosome
4: **repeat**
5: Select parents
6: Crossover pairs of parents
7: mutate offspring
8: Evaluate new individuals
9: Select individuals for the next generation
10: **until** Stopping criterion is not satisfied

The main steps of the standard GA are given in Algorithm 6. The algorithm starts with a population of individuals (chromosomes) which is randomly generated (line 2). Then, the fitness of each chromosome is evaluated to see how good it is at solving the problem at hand (line 3). Then, The transition from a generation to another one is achieved by repeatedly applying these three operators (lines from 5 to 9): select parents, crossover pairs of parents, mutate offspring. Finally, the algorithm terminates when a stopping criterion (a maximum number of generations has been produced, or a satisfactory fitness level has been reached) is satisfied.

Hifi *et al.* [29] have used this method for cutting (non)identical circle from a fixed-dimension rectangle. The initial population contains solutions represented by possible positions for circles. Part of the solution has been obtained according to the criterion of the best local position of the circle to be cut, and the remaining solutions have been generated randomly. Selection, crossover and mutation procedures have been used.

B- Particle swarm optimization

Particle swarm optimization (noted PSO) is a meta-heuristic based on a population (cf. Kennedy *et al.* [42]). It can efficiently explore several spaces of candidate solutions during the search process. Like other meta-heuristics, PSO does not guarantee the optimality

of the solutions achieved, but it generally ensures an experimental convergence toward high-quality solutions. Also note that the method is interesting due to its simplicity and the number of parameters that are used in contrast to other evolutionary methods.

Each particle in PSO characterizes a solution of the given problem. It is represented by its current position $\vec{X}$ on the search space, its best-visited position $\vec{P}_{best}$ during the search and its velocity $\vec{v}$. Furthermore, each particle shares information with the other particles of the population (swarm) and updates its current positions by using a simple formula that acts on the velocity; it is defined as follows:

$$\overrightarrow{v_i}^t = \omega \times \overrightarrow{v_i}^{t-1} + c_1 \times \upsilon \times \left[\overrightarrow{P}_{best} - \overrightarrow{x_i}^{t-1}\right] + c_2 \times \nu \times \left[\overrightarrow{G}_{best} - \overrightarrow{x_i}^{t-1}\right] \tag{3.5}$$

and

$$\overrightarrow{x_i}^t = \overrightarrow{x_i}^{t-1} + \overrightarrow{v_i}^t, \tag{3.6}$$

where Eq. (3.5) acts on the velocity and Eq. (3.6) acts on the particle's positions.

One can observe that Eq. ((3.5)) is composed of three parts:

(i) $\omega \times \overrightarrow{v_i}^{t-1}$: it represents the i-th particle's velocity at iteration $(t-1)$, where ω denotes the inertia weight (introduced by Shi *et al.* [64]). Such a weight tries to control the magnitude of the "old velocity" and usually takes its values in the interval $[0.4, 0.9]$.

(ii) $\left[\overrightarrow{P}_{best} - \overrightarrow{x_i}^{t-1}\right]$: it represents a natural tendency of a particle to return to its best position, namely p_{Best}.

(iii) $\left[\overrightarrow{G}_{best} - \overrightarrow{x_i}^{t-1}\right]$: it represents the tendency of a particle to follow the best position, namely $\overrightarrow{G}_{best}$, reached by any member belonging to its neighborhood.

The parameters, c_1 and c_2, represent the cognitive and social factors, respectively, where $c_1 + c_2 \leq 4$, whereas both υ and ν are randomly generated in the interval $[0, 1]$; these values are used to determine the degree of influence of $\overrightarrow{P}_{best}$ and $\overrightarrow{G}_{best}$, respectively.

The main steps of the standard PSO are given in Algorithm 7. It starts with a population of size K. The initialization step (line 2) searches for the position $\vec{X}^i$ of each particle i which are randomly generated or by using a simple heuristic and $\vec{P}^i_{best}$ of each particle i is setting equal to the value of the current position $\vec{X}^i$ and the velocity vector is setting equal to zero. The best position $\vec{G}^i_{best}$ is setting equal to the best solution of the starting population (line 4). Therefore, the fitness of each particle is evaluated, where the best fitness reached is stored in $\overrightarrow{P}_{best}$ (line 7). Furthere, both velocity and position of each particle i are updated (lines 15,16). The algorithm exits with the best position visited which is considered as the best solution to the problem. Figure (3.2) illustrates the behavior of the PSO algorithm where the Particles do not stop searching

Algorithm 7 – A standard version of PSO algorithm

Input. A population of size K and the different parameters used.

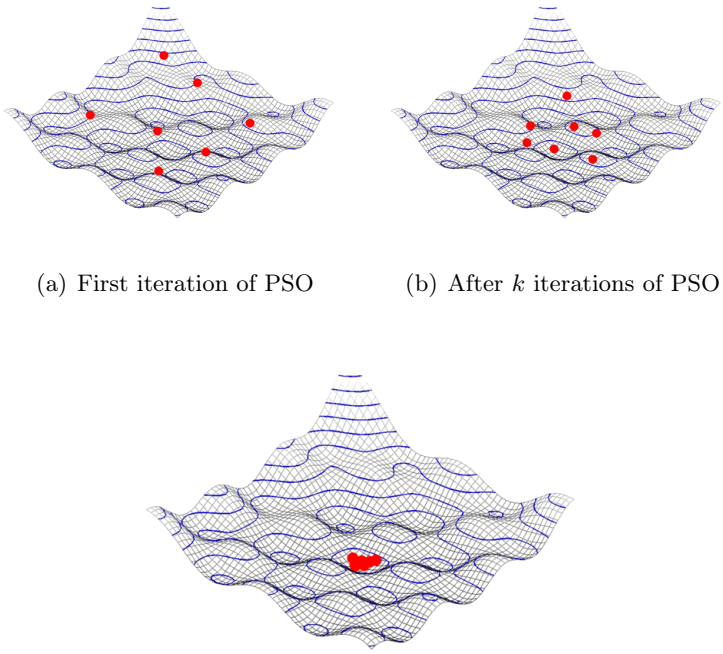
Output. A best particle $\overrightarrow{G}_{best}$.

Initialization step.
Generate the position $\vec{X}^i$ of each particle.
Set $\vec{P}^i_{best} = \vec{X}^i$ and let $\vec{V}^i$ be the velocity vector.
Choose the best $\vec{P}^i_{best}$ as $\overrightarrow{G}_{best}$
Iterative step
while (the runtime limit is not performed) **do**
  for (each particle $i \in K$) **do**
    Compute its fitness function $F_{\vec{X}^i}$
    if ($F_{\vec{X}^i} < F_{\vec{P}^{\ i}_{best}}$) **then**
      $\vec{P}^i_{best} = \vec{X}^i$
    end if
  end for
  Choose the particle realizing the best value $F_{\vec{P}^{\ i}_{best}}$ as $\overrightarrow{G}_{best}$
  for (each particle $i \in K$) **do**
    Compute the particle's velocity according to Eq. (3.5)
    Update the particle's position according to Eq. (3.6)
  end for
end while
Return the best solution achieving the best global solution value $\overrightarrow{G}_{best}$.

until $\vec{X}^i = \vec{P}^i_{best} = \overrightarrow{G}_{best}$ and their velocity becomes zero. This condition is necessary for the convergence to an equilibrium point.

In the global version of PSO, the neighborhood consists of the whole population (fully connected PSO (cf. figure 3.3)) and $\overrightarrow{G}_{best}$ represents the best particle. Note that, the disadvantage of this version occurs in the case when it leads the swarm to be trapped in a local minimum, because the whole swarm takes its social knowledge from a unique source represented by $\overrightarrow{G}^k_{best}$. To avoid this problem, a version of PSO has been proposed in the literature (under the name of "local version $\overrightarrow{L}_{best}$") in which each particle receives information from a subset of k neighbors instead of the whole population (cf. Kennedy *et al.* [43, 41], Suganthan [68]).

Figure (3.3) represents several neighborhood structures proposed in the literature. Among these structures, we can mention (i) Ring structure: each particle is bound to two neighbors, one on each side, (ii) Von Neumann structure: each particle has four neighbors and (iii) Random structure: each particle has k randomly chosen neighbors.



(a) First iteration of PSO (b) After k iterations of PSO

(c) After n iterations of PSO

Figure 3.2: Behaviour of the PSO algorithm

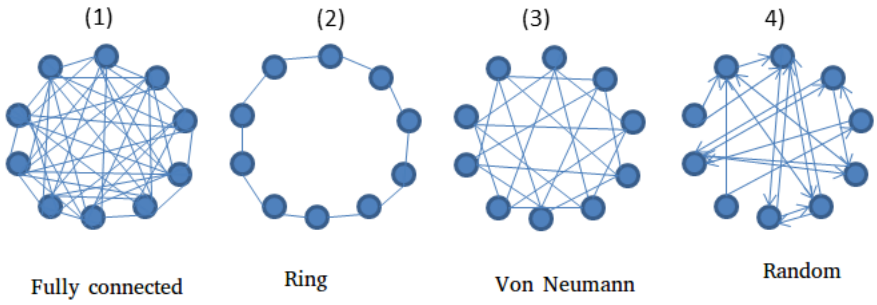


Figure 3.3: Graphical representation of the neighbourhood structure

3.3.3 Methods based on simulation

In the literature, some researchers have proposed approaches which simulate certain physical phenomena for solving circle/sphere packing problems. For example, a simulation of elastic discs has been proposed by Wang *et al.* [72], a Billiards Simulation has been used by Graham *et al.* [25] and Lubachevsky and Graham [53].

Billiard Simulation is a stochastic method that simulates the idealized movement of billiard balls inside a domain. This method has been used only to solve the identical circles packing problem in a circular container and in a square, whereas elastic discs simulation is more efficient; it is used for solving identical/non identical circle /sphere packing in different containers.

In what follows, we will give more detail on the elastic discs simulation since we have used it as a sub-procedure in some of the approaches that we have developed.

Elastic discs simulation

This method has been proposed by Wang *et al.* [72] (named "Quasi-Physical" procedure, denoted by QPP) in order to solve the circle / sphere packing problem into a circular / spherical container of fixed dimensions. The main principle of such a method can be summarized as follows:

The container is considered as a "fixed elastic container" and all the spheres as "smooth elastic solids" which are forced to fit inside the container. According to the elasticity mechanism, there exist some conjugated extrusion elastic force whenever some spheres are deformed by other ones or by the container. In this case, a series of complicated distortions would occur due to these elastic forces. During the search process, the potential energy function (the overlapping gap either between the deformed spheres or that realized between spheres and the container) will be reduced to zero. Reducing such deformations to zero induces the converge of the procedure to a local optimum.

In order to present the potential energy function, the amount of overlapping depths between two smooth elastic spheres and between a smooth elastic sphere and the container must be calculated as its represented in Table 3.2.

overlapping depth Between	Open container	Spherical Container
sphere and container	$\delta_{i,x} = \max\left\{0, r_i + \lvert x_i\rvert - \frac{1}{2}L\right\}$ $\delta_{i,y} = \max\left\{0, r_i + \lvert y_i\rvert - \frac{1}{2}H\right\}$ $\delta_{i,z} = \max\left\{0, r_i + \lvert z_i\rvert - \frac{1}{2}W\right\}$	$\delta_{0,i} = \max\{0, \Delta(i,0) + r_i - R\}$
sphere and sphere	$\delta_{i,j} = \max\{0, r_i + rj - \Delta(i,j)\}$	$\delta_{i,j} = \max\{0, r_i + rj - \Delta(i,j)\}$

Table 3.2: Overlapping depths for both problems: between the positioned spheres and between spheres and the container

In table 3.2, $\Delta(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}, i, j \in N, i \neq j$ and $\Delta(i,0) = \sqrt{x_i^2 + y_i^2 + z_i^2}, i \in N$ are the euclidean distance between the center of two spheres and between the i^{th} sphere and the container.

According to both terms of overlapping calculated in table 3.2, the elastic force acting on the i^{th} sphere and the potential energy function of both problems (spherical/open

container) are calculated in Table 3.3:

	Open container	Spherical Container
Elastic force acting on the i^{th} sphere	$U_i = \sum_{j \neq i}^{N} \delta_{ij}^2 + (\delta_{x_i 0} + \delta_{yi0} + \delta_{zi0})$	$U_i = \sum_{j=0}^{N} \delta_{ij}^2$
The potential energy function	$E(\overrightarrow{X}) = \sum_{i=1}^{N} U_i$	$E(\overrightarrow{X}) = \sum_{i=i}^{N} U_i$

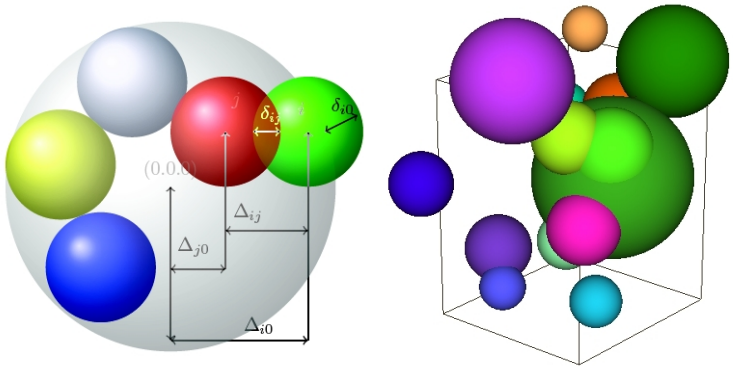
Table 3.3: Extrusion elastic potential force acting on the i^{th} sphere and potential energy functions

According to the functions presented in table 3.3, one can observe what follows:

1. First, the elastic force proportional to the overlapping depth between any two spheres i and j. When i and j are identical spheres, they bounce away each other and move a same distance under the elastic force. For two elastic spheres in different sizes, the deformation of the larger one is generally less than that of the smaller one when they overlap each other and the elastic potential force U_i can be used to measure the degree of pain of the i^{th} (Wang *et al.* [72]).

2. Second, a solution is considered as a feasible one **if and only if** $E(\overrightarrow{X}) = 0$.

Figure 3.4(a) illustrates an non-feasible solution (configuration) for packing into (i) a spherical container and (ii) an open container. A solution (configuration) is represented by a vector $\overrightarrow{X} = (x_i, y_i, z_i, ..., x_n, y_n, z_n)$ which contains the coordinates of the n packed spheres.



(a) Overlapping in the case of packing in a spherical container

(b) Overlapping in the case of packing in an open container

Figure 3.4: Physical movement according to elastic forces

As shown in figure 3.4(a), the sphere i overlaps both the container and the sphere j. By the action of the elastic forces, the sphere j will move alone the direction of from i to j. The magnitude of the movement depends on the overlapping depth δ_{ij}. If the sphere i keeps action-less, then sphere j moves its depth δ_{ij} until it does not overlapped with sphere i. if sphere j keeps action-less, then the container makes sphere i move δ_{i0} along the direction of from i to the center of the container (indexed 0) and sphere j makes sphere i move its depth δ_{ij} along the direction of from j to i. If the overlapping depth is considered as a vector, then the magnitude of the movement is the sum of all its overlapping depth vectors and the direction is the direction of the vector sum of all elastic forces acted on a given sphere.

The magnitude of the movement can be calculated by the projection of all overlapping depth vectors in axes x, y and z as follows:

Projection in	Moving distance produced by the sphere j	Moving distance produced by the container
Axis X	$d_{x_j} = \frac{x_i - x_j}{\Delta_{ij}}\delta_{ij}$	$d_{x_i} = \frac{-x_i}{\Delta_{i0}}\delta_{i0}$
Axis Y	$d_{y_j} = \frac{y_i - y_j}{\Delta_{ij}}\delta_{ij}$	$d_{y_i} = \frac{-y_i}{\Delta_{i0}}\delta_{i0}$
Axis Z	$d_{z_j} = \frac{z_i - z_j}{\Delta_{ij}}\delta_{ij}$	$d_{z_i} = \frac{-z_i}{\Delta_{i0}}$

Table 3.4: Calculate the moving distance by the projection overlapping depth on axes x, y and z

Therefore, the new position of sphere i $(\overline{x_i}, \overline{y_i}, \overline{z_i})$ is calculated according to the equation (3.7).

$$\overline{x_i} = x_i + d_{x_i} + d_{x_j}, \quad \overline{y_i} = y_i + d_{y_i} + d_{y_j}, \quad \overline{z_i} = z_i + d_{z_i} + d_{z_j} \tag{3.7}$$

Algorithm 8 Quasi-physical QPP

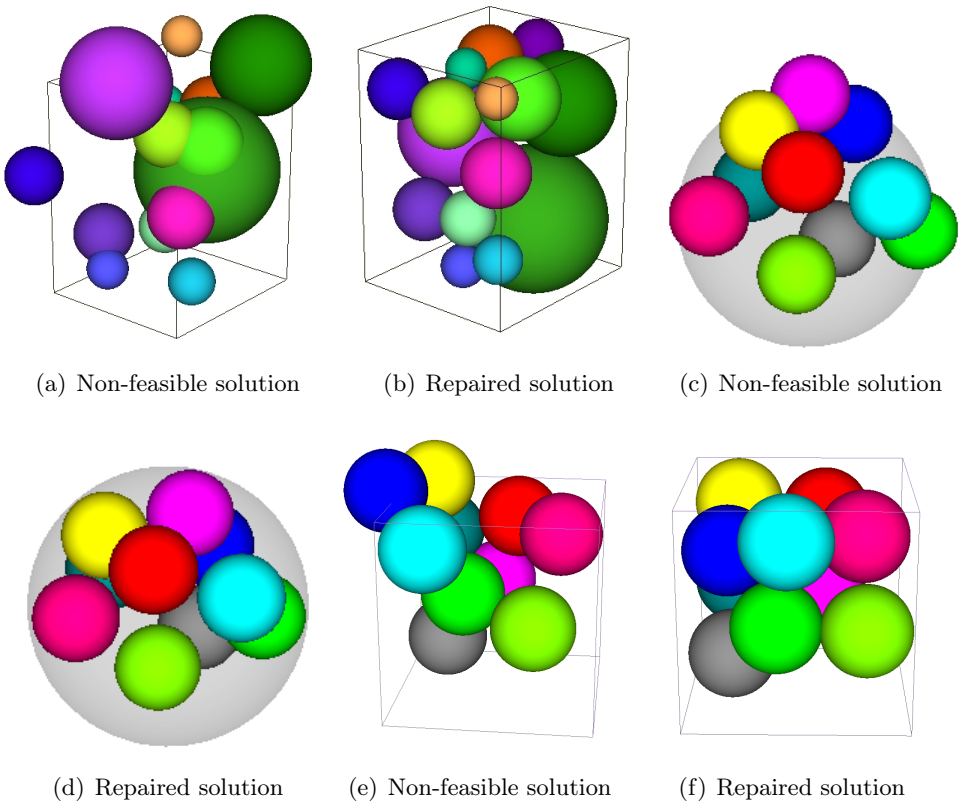
Input A non-feasible solution $\vec{X} = (x_1, y_1, z_1, x_2, y_2, z_2 ... x_n, y_n, z_n)$, ε_1, ε_2,ε_3, $h = 1$.

Output A feasible solution or a trapped local minimum.

1: **Initialization step**
2: **Calculate** $E(\overrightarrow{X})$ **by using the equations presented in tables (3.2 and 3.3)**
3: **if** $E(\overrightarrow{X}) < \varepsilon_1$ **then**
4: this solution is feasible, return $\vec{X}$
5: **end if**
6: Iterative step
7: **while** $\left(E(\overrightarrow{X}) > \varepsilon_1 \text{ et } h > \varepsilon_2\right)$ **do**
8: $\vec{X_1} = \vec{X}$
9: **for** each sphere in $\vec{X_1}$ **do**
10: Calculate the elastic force by using equations presented in tables (3.2 and 3.3)
11: Move it into a next postion by using equation 3.7 with step size h
12: **end for**
13: **Calculate** $E(\overrightarrow{X_1})$ **by using the equations presented in tables (3.2 and 3.3)**
14: **if** $E(\overrightarrow{X_1}) < E(\overrightarrow{X})$ **then**
15: $E(\overrightarrow{X}) = E(\overrightarrow{X_1})$, $\vec{X} = \vec{X_1}$
16: **end if**
17: **otherwise**: decrease the step size $h = \varepsilon_3 * h$
18: end otherwise
19: **end while**
20: Return feasible solution $\vec{X} = (x_1, y_1, z_1, x_2, y_2, z_2 ... x_n, y_n, z_n)$ or a trapped local minimum

Algorithm 8 describes the main steps of Quasi-Physical (noted QPP). It receives as an input a configuration $(\vec{X} = (x_1, y_1, z_1, x_2, y_2, z_2 ... x_n, y_n, z_n))$. The four parameters $\varepsilon_1, \varepsilon_2, \varepsilon_3$ and h have the following roles: ε_1 is the accuracy of the computation which we are willing to accept the configuration as a feasible solution of the packing problem. ε_2 is the value (h) with which the algorithm stops in case it does not find a feasible solution. ε_2 is a multiplication factor used to decrease the step size h. h has the initial value 1.

During the initialization step (line 2), if there is no overlap, the algorithm returns a feasible solution. Otherwise, the iteration step starts at line 7. Two loops are performed alternately: the "while" loop controls the stopping criterion of the algorithm, and the "for" loop computes the overlapping quantity and measures the elastic forces by specifying the magnitude of the movement of each sphere and its direction. At line (11), a new configuration is produced. If it is better than the previous one, it replaces it (line 15). Otherwise, h is decreased by multiplication by ε_3 (line 17). Finally, QPP algorithm stops as soon as it has found a feasible solution or if h becomes inferior to ε_2. In the latter case, the algorithm is trapped in a local optimum. Figure (3.5) illustrates the behavior of QPP algorithm (8).



(a) Non-feasible solution (b) Repaired solution (c) Non-feasible solution

(d) Repaired solution (e) Non-feasible solution (f) Repaired solution

Figure 3.5: Behavior of QPP algorithm

3.4 Conclusion

In this chapter, we have presented other variants belonging to the sphere packing family, such as packing spheres into a cuboid, a cube, a cylinder, a pyramid, a regular tetrahedron. We have also presented some of the well-known solution methods developed for solving some sphere/circle packing problems. Those problems belong to the NP-hard class, where no algorithm can solve them optimally in a polynomial time. Consequently, the approaches available in the literature (specific heuristics and meta-heuristics, methods by experimental simulation), used for tackling such problems, have been presented in this chapter.

Chapter 4

A dichotomous search-based heuristic for the three-dimensional sphere packing problem

Contents

In this chapter, the three-dimensional sphere packing problem is solved by using a dichotomous search-based heuristic. We recall that an instance of the problem is defined by a set N of n unequal spheres and an open container of fixed width and height and, unlimited length. Each sphere is characterized by its radius and the aim of the problem is to optimize the length of the open container containing all spheres without overlapping. The proposed method is based upon beam search, in which three complementary phases are combined: (i) a greedy selection phase which determines a series of eligible search subspace, (ii) a truncated tree search, using a width-beam search, that explores some promising paths, and (iii) a dichotomous search that diversifies the search. The performance of the proposed method is evaluated on benchmark instances taken from the literature where its obtained results are compared to those reached by some recent methods of the literature.

4.1 Introduction

Cutting and Packing problems (CP) is a family of natural combinatorial optimization problems, admitted in numerous real-world applications from industrial engineering, logistics, manufacturing, production process, automated planning, etc. One of the more recent paper addressing an optimization with a packing problem is due to Sutou and Dai [69], where the unequal sphere problem has been used for tackling an application of the automated radiosurgical treatment planning. Wang [73] has also considered sphere packing problems as an optimization tool for the radiosurgical treatment planning. Other problems of the CP family have been described and redefined in Wascher *et al.* [74] where an instance of these problems can be defined by a set of predetermined items to be packed in one or many larger containers (objects) so as to minimize the unused area / space or in some cases to maximize a utility function. Furthermore, the items are bounded by their dimensions (rectangular, circular, or irregular) and the objects can be bounded (rectangular, circular, ...) or unbounded (strips / parallelepipeds, ...).

This chapter deals with the problem of packing spheres (called items) in a container (parallelepiped or object) with unlimited length. The problem is known as the *Three-Dimensional Sphere Packing Problem* (3DSPP). An instance of 3DSPP is defined by a set N of n unequal items and an object $\mathcal{P}$ of fixed width W and height H and, unlimited length (for the rest of the paper, the length representing the variation of P's length is denoted L). Moreover, each item $i \in N = \{1, \ldots, n\}$ is characterized by its radius r_i. The goal of the problem is to optimize the length L of the object $\mathcal{P}$ such that all items of N are packed (positioned) in the target object, without overlapping.

The 3DSPP may be formulated as follows:

$$\min \quad L \tag{4.1}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \geq (r_i + r_j)^2 \qquad \forall (i,j) \in N^2,\ i < j \tag{4.2}$$

$$r_i \leq x_i \leq L - r_i \qquad \forall i \in N \tag{4.3}$$

$$r_i \leq y_i \leq H - r_i \qquad \forall i \in N \tag{4.4}$$

$$r_i \leq z_i \leq W - r_i \qquad \forall i \in N \tag{4.5}$$

$$\underline{L} \leq L \leq \overline{L} \tag{4.6}$$

where the objective function (4.1) minimizes the length of the object $\mathcal{P}$ containing all the items of N, Equation (4.2) ensures the non-overlap constraint of any pair of distinct items (i, j) of N; that is, the distance between the centers of both items which must be greater than or equal to the sum of their radii and, Equations (4.3)-(4.5) ensure that all items of N belong to the target object $\mathcal{P}$ of dimensions (L, W, H). Note that since the aim of the problem is to find the smallest length of the object containing all items of N, then it is easy to start any method trying to solve the above problem by a trivial value representing

the sum of the spheres' area affected to $\underline{L}$ (Equation (4.6)) and a quick (feasible) solution value with a greedy algorithm affected to $\overline{L}$.

In this chapter, we propose a three phase solution procedure that combines the width-beam search and a greedy selection procedure for approximately solving the 3DSPP. The last two phases are embedded in a dichotomous procedure that is used for searching the best length $L^{\star}$ for the object. Note that the proposed algorithm can be viewed as an adaptation of the method proposed in Hifi *et al.* [32] for solving the two-staged two-dimensional cutting problem and later in Akeb and Hifi [4] for solving another variant of CP family. The novelty in this approach is based upon (i) a new representation used for defining a restricted space search; that is, a structured set based on target eligible positions which serves to limit the search process without degrading the quality of solutions and (ii) a nice dichotomous search that tries to avoid premature convergence toward local stagnant solutions whenever the width-beam search is applied.

The rest of the chapter is organized as follows. The problem representation is discussed in Section 4.2.1. Section 4.2.2 exposes the principle of the greedy selection phase which serves to determine a subset of eligible positions employed for packing the predefined items in the target object. Section 4.2.3 presents a truncated tree-search using a width beam search phase that serves to explore some promising paths. Section 4.2.4 exposes the dichotomous search where the first two phases are embedded in an interval search in order to diversify the search process. Section 4.3 proposes an experimental part in which the performance of the proposed algorithm is given: its provided results are compared to those reached by some best algorithms of the literature. Finally, Section 4.4 concludes by summarizing the contribution of this paper.

4.2 Tackling 3DSPP with a dichotomous search

Herein, the problem representation and strategies used by the proposed algorithm are discussed. First, section 4.2.1 describes how an item can be represented in a space and which eligible positions may be chosen for assigning items of N to the target object $\mathcal{P}$. Second, section 4.2.2 describes the principle of the simple constructive procedure which tries to guaranty a feasible packing of all items of N for the 3DSPP according to the set of eligible positions. Such a simple constructive procedure is based on the minimum distance between items and the edges of the objet $\mathcal{P}$ that serves to create a set of eligible positions for the current item to pack. Third and last, section 4.2.3 discusses the principle of the proposed algorithm and its main steps.

4.2.1 Representation of 3DSPP

In this section, we present the principle of the *local strategy* used for filtering the huge number of possible positions when an item will be assigned to the object $\mathcal{P}$. The local

strategy is based on the simple Greedy Principle (called GP) where the minimum distance position is favored for packing a predefined item. GP is then used as an evaluation operator for finding a subset of eligible positions of the next item to pack. Furthermore, the GP can also be used either as a complete solution procedure that provides a final feasible solution for the 3DSPP or as a greedy strategy in order to complete a partial solution.

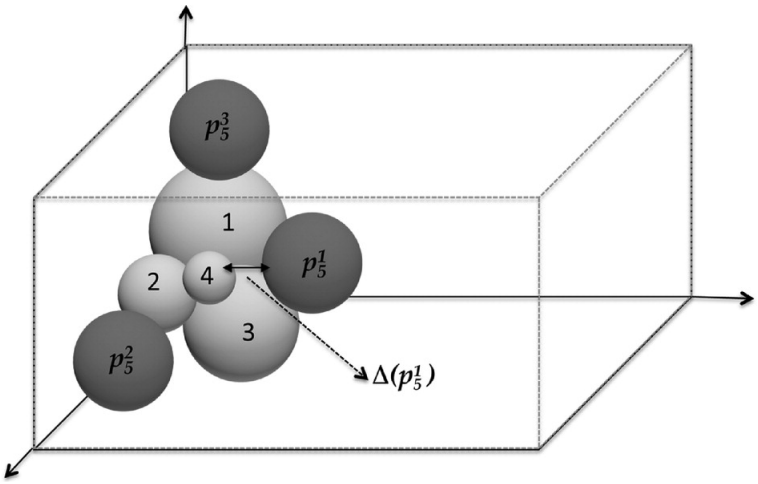


Figure 4.1: Illustration of the mechanism used for computing eligible positions.

For the rest of the paper, the following notations are introduced:

- The bottom-left-depth corner of $\mathcal{P}$ is positioned at $(0, 0, 0)$ and $\mathcal{P}$ is characterized by a set formed with six labels (namely faces): $\mathbb{F} = \{\text{left}, \text{top}, \text{right}, \text{bottom}, \text{depth}, \text{front}\}$. Then, $\mathcal{P}$ is represented in the Euclidian space, as illustrated in Figure 4.1.

- The center of the i-th item belonging to N is positioned at (x_i, y_i, z_i).

- The distance, namely $\delta_{i,j}$, between two different items i and j belonging to N is computed as follows:

$$\delta_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} - (r_i + r_j), \quad \forall (i, j) \in N^2 \tag{4.7}$$

Observe that positioning any item i of N into $\mathcal{P}$ induces to define a distance between an item belonging to N and faces of $\mathbb{F}$. Indeed, for instance, assigning an item $i \in N$ to an eligible position of $\mathcal{P}$ while respecting the non-overlap of the left-face of $\mathcal{P}$ requires to satisfy the following distance: $\delta_{i,\text{left}} = x_i - r_i$. For all faces of $\mathcal{P}$, Table 4.1 reports the distance to be satisfied whenever a preselected item i of N is assigned to an eligible position of $\mathcal{P}$.

Table 4.1: The distance between an item i and a face f

f	$\delta_{i,f} \mid i \in N,\ f \in \mathbb{F}$
left	$x_i - r_i$
bottom	$y_i - r_i$
depth	$z_i - r_i$
right	$L - x_i - r_i$
top	$H - y_i - r_i$
front	$W - z_i - r_i$

4.2.2 Defining eligible positions

It is well-known that tailored heuristics are mainly based on the strategies which are able to guide well the search process. These strategies may be use some selection criteria in order to provide either partial or final solutions for the problem to solve. Herein, we consider a simple greedy principle (GP) which is based on searching the position realizing the minimum distance position (MLDP) between items and feces. In fact, GP is used as a selection criterium for defining a set of eligible positions to assign to the predefined item i (not already positioned) among all eligible positions representing the set P_{I_i}.

In what follows, we assume that the (center of the) first item $i = 1$ of N is positioned at the position (r_1, r_1, r_1) and $\forall i \in N,\ i \geq 2$, the following notations are considered:

- I_i denotes the set of items of N already positioned in the current object $\mathcal{P}$.
- $\overline{I_i}$ contains the items of N which are not yet assigned to $\mathcal{P}$.
- P_{I_i} denotes the set of distinct eligible positions for the next item i to pack given the set of packed items I_i.
- An eligible position $p_{i+1} \in P_{I_i}$ (for the item i) is determined by using three elements $e_1,\ e_2$ and e_3 where an element is either an item of N already positioned (representing the set I_i) or one of the six faces belonging to $\mathbb{F}$.
- $\mathcal{T}_{p_{i+1}}$ represents the set composed of the three elements $e_1,\ e_2$ and e_3.

Figure 4.1 illustrates the mechanism used by the GP strategy on a small example. We assume that the first four items are already positioned in the object $\mathcal{P}$; then, there are three eligible positions that emerge for the next item 5 to pack. Following the above notations, $I_4 = \{1, 2, 3, 4\}$ and $P_{I_4} = \{p_5^j,\ j = 1, \ldots, 3\}$. First, the position p_5^1 touches both items 1 and 3 and, the "bottom" face of $\mathcal{P}$. Second, the position p_5^2 is obtained by using the item 2 and both faces "left" and "bottom" of $\mathcal{P}$. Third and last, the position p_5^3 is computed by using the item 1 and both faces "left" and "depth" of $\mathcal{P}$. Finally, it follows that $\mathcal{T}_{p_5^1} = \{1, 2, \text{bottom}\},\ \mathcal{T}_{p_5^2} = \{2, \text{left}, \text{bottom}\}$ and $\mathcal{T}_{p_5^3} = \{1, \text{left}, \text{depth}\}$.

We can observe that the item 4 is positioned around the three items 1, 2 and 3 that means $\mathcal{T}_4 = \{1, 2, 3\}$. For the next item 5 to pack, its coordinates are computed by using both items 1 and 3 and one of the faces (the "bottom" in this case) of $\mathcal{P}$ that gives $\mathcal{T}_5 = \{1, 3, \text{bottom}\}$. We recall that the objective of the problem is to minimize the length of the target object $\mathcal{P}$. It means that the right face of $\mathcal{P}$ can be omitted and only the five faces can be considered when optimizing the length of the target $\mathcal{P}$. Hence, all eligible positions may be obtained by using the fifth faces, the positioned items and the current item to pack.

Moreover, GP works as follows: the value corresponding to the $(i+1)$-th item to pack, when positioned at the eligible position $p^k_{i+1} \in P_{I_i}$, is computed as follows:

$$\Delta(p^k_{i+1}) = \min_{j \in I_i \cup \mathbb{F}' \setminus \mathcal{T}_{p^k_{i+1}}} \delta_{(i+1,j)} \tag{4.8}$$

where $\mathbb{F}' = \mathbb{F} \setminus \{\text{right}\}$.

Finally, when GP is used, then it starts by positioning the first item $i = 1$ at the bottom-left-depth position, i.e., at the position (r_1, r_1, r_1), while the remaining $n-1$ items are successively positioned according to the minimum distance rule (cf., Equation (4.8)). As illustrated in Figure 4.1, the item 5 will be placed at position p^1_5 because its corresponding distance realizes the minimum value (of course, this choice may also be modified along the execution of the algorithm).

Note that one can use GP as a greedy procedure for searching a feasible solution for the 3DSPP. Indeed, by setting the starting length of the target object $\mathcal{P}$ to ∞ (i.e., $L_{\mathcal{P}} = \infty$) and by applying successively GP on the rest of the non packed items, GP terminates by building a feasible solution with length L_U. The provided length L_U can also be considered as an upper bound for starting the proposed algorithm.

4.2.3 A width-beam search heuristic for the 3DSPP

In this section, we first describe the principle of the standard beam search (Section 4.2.3.1). Second and last, we show how beam search can be adapted for solving the 3DSPP (Section 4.2.3.2), especially when using a width-beam almost of the standard beam.

4.2.3.1 A standard beam search

Beam Search (BS) has been first proposed by Ow and Morton [62] for tackling the scheduling problem and it has since been successfully applied to many other combinatorial optimization problems (some adaptations can be found in Della Croce *et al.* [19], Hifi *et al.* [30, 32, 34] and, Yavuza [76]). Such an approach can be viewed as a truncated tree search procedure where its objective is to avoid exhaustive search by performing a partial enumeration of the solution space.

At each level of the developed tree, only a subset of nodes (called the *set of elite nodes*) are selected for further branching and the other nodes are discarded, where no backtracking is performed. For each level, the cardinality of the elite nodes to be investigated is fixed to ω that is called the *beam width*. Generally, these selected ω nodes represent those having a high potential to lead the best solutions for the treated problem. Furthermore, each node is assessed via an *evaluation function* whose role is to provide a promising separation mechanism of the nodes of each level of the developed tree.

Algorithm 9 . A standard beam search.

1. **Initialization Step.**
   (a) Let ω be the beam width.
   (b) Set $B = \{B_0\}$ and $B_\omega = \emptyset$, where B is the set of nodes to be investigated, and B_ω the set of nodes branched out of the nodes in B.
   (c) If an initial feasible solution is available, set $z^\star$ to its objective function value; otherwise, set $z^\star = +\infty$ (minimization problem)
2. **Iterative Step.**
   Repeat
   (a) Choose a node $\eta \in B$; branch out η; remove η from B and insert the created nodes into B_ω.
   (b) If a node η of B_ω is a leaf, then
      i. compute its objective function value z_η;
      ii. if $z_\eta > z^\star$, update $z^\star$ and the incumbent solution;
      iii. remove η from B_ω.
   (c) Assess the potential of each node of B_ω using an evaluation function.
   (d) Rank the nodes of B_ω in a decreasing order of their values.
   (e) Insert the $\min\{\omega, |B_\omega|\}$ best nodes (called elite nodes) of B_ω into B; and set $B_\omega = \emptyset$.

   Until $B = \emptyset$.

Algorithm 9 describes the main steps of the standard beam search which is characterized by a beam width ω used for filtering the set of offspring nodes B_ω. A node corresponds to a partial feasible solution and the set B of current nodes is initialized to the root node B_0 whereas B_ω containing the offspring nodes is initialized to the empty set. Each node of B generates a set of offspring nodes, and moves them to B_ω. If a node η of B_ω is a leaf (i.e, no further branching is possible out of η), then its objective function value z_η is computed and compared to $z^\star$. If $z_\eta > z^\star$, then the incumbent solution is set to the leaf node; $z^\star$ is then updated: $z^\star = z_\eta$; and η is removed from B_ω. All nodes belonging to B_ω are assessed using an evaluation function, and then ranked in decreasing order of their values. The first ω nodes of B_ω are then considered as the elite nodes which are appended to the set B, whereas the remaining nodes of B_ω are discarded and B_ω is reset to the empty set. This process is iterated until no further branching is possible, i.e., until $B = \emptyset$.

4.2.3.2 Adaptation of the beam-search for the 3DSPP

Applying BS to 3DSPP requires defining the nodes of the tree and the branching mechanism out of the nodes of B. Herein, a node η_i is represented by the pair of subsets:

1. The first subset I_i containing all items assigned to the target object $\mathcal{P}$ and,
2. The complementary subset $\overline{I}_i$ containing the unassigned items.

The first subset represents a *partial solution* realized when assigning all the items to the object $\mathcal{P}$ and the second subset serves to provide a *complementary solution*. Such a complementary solution is computed by applying the MLDP used as greedy procedure (as discussed in Section 4.2.2).

Moreover, branching out of a node η_i is equivalent to create at most $|P_{I_i}|$ branches emanating out of the current node (related to the eligible positions as described in Section 4.2.2). Each resulting node corresponds to packing the subset of items I_i and assigning to the current item i a favorite eligible position. Moreover, each of these created nodes will be represented by a pair of two complementary subsets of items of N.

As mentioned above, because of the huge number of feasible positions that a predefined item may generate, we preferred the use of the width-beam search almost of the standard beam search. Indeed, as we shall see in Section 4.2.4, the proposed algorithm uses a dichotomous search according to a series of values assigned to ω (the beam width). Therefore, in order to try the exploration of a maximum number of promising paths, the beam search applies the width-search where all nodes belonging to the same level are simultaneously evaluated following an estimator operator and only the best ones are selected for the rest of the search.

The aforementioned process is described by Algorithm 10. The adapted algorithm works according to a given node, namely η^ℓ, taken at the level ℓ of the developed tree. Thereafter, the initialization step (Lines 1 to 3) is applied for starting the set B containing the best provided nodes regarding the starting node η^ℓ and, the initialization of the variable `feasible` to false. This variable controls the (un)feasibility of the series of the solutions builded. The main loop (lines 7) starts by choosing the best eligible positions for each node belonging to B. These positions are computed by using GP's selection (cf., Section 4.2.2). Second, all created nodes are stored in a provisional set B_ω where the potential of each of these nodes are evaluated according to the final solution provided by iteratively applying GP as a heuristic (cf. as discussed in the last paragraph of Section 4.2.2). Thereafter, for each final solution (either feasible or unfeasible for the target object $\mathcal{P}$), the potential of a node $\eta \in B_\omega$ is represented by the *density of the positioned items* in $\mathcal{P}$. Whenever one of these constructed solutions provides a feasible solution (line 10), i.e., all items are positioned in the target object $\mathcal{P}$, then the algorithm stops with a feasible solution (i.e., setting the variable `feasible` to true). Otherwise, the set B of the best nodes is updated (line 11) by the ω nodes which realizing the highest densities

and the current level of the developed tree is incremented. Finally, the process is iterated until either B is reduced to an empty set or when the fixed runtime limit is performed.

4.2.4 Using a dichotomous search

The proposed algorithm applies a series of BS for a predefined interval search. First, the interval search starts by $[\underline{L}, \overline{L}]$, where $\underline{L}$ denotes a lower bound for the 3DSPP and $\overline{L}$ its upper bound (in the case of a feasible solution exists, its objective value is assigned to $\overline{L}$). Second, for each fixed interval, BS tries to construct the best feasible solution by packing all the items into the current target object; that is, $(L^\star, W, H)$, where $L^\star \in [\underline{L}, \overline{L}]$.

The main steps of the dichotomous search are summarized in Algorithm 11. First, its starts by defining the initial interval $[\underline{L}, \overline{L}]$ where the upper bound $\overline{L}$ is obtained by applying GP as a heuristic on the open object, i.e., (∞, W, H). The main loop repeat (cf., lines 7 - 12) of the dichotomous procedure serves to explore a series of neighborhoods depending on the values of ω. At line 8, a new target upper bound is computed, namely $L^\star = (\overline{L} + \underline{L})/2$. Line 9 generates the initial node positioned at the bottom-left-depth corner (in the position (r_1, r_1, r_1)) and creates its corresponding sets I_i, $\overline{I}_i$ and P_I^1 (as discussed in Section 4.2.2). At line 10, BS is called with the target value of the object $(L^\star, W, H)$ and the created sets reached at the next step. Line 11 serves to update the interval search where its upper bound is updated whenever a feasible solution is obtained, the lower bound is updated otherwise. Thereafter, the process is iterated till the gap between both lower and upper bounds becomes closest to a certain tolerance, namely α.

Algorithm 10 . Adaptation of the Beam Search for the 3DSPP: BS

Input. A node η^ℓ.

Output. `feasible` // setting equal to `true` whenever a feasible packing is reached, `false` otherwise

1: **Initialization Step.**
2: Let ω be a predefined beam width.
3: Set $B = \{\eta^\ell\}$, where η^ℓ denotes the input node associated to the ℓ-th level.
4: Set the variable `feasible` to `false` /* no feasible solution at hand */
5: **Iterative Step.**
6: **while** $\Big((B \neq \emptyset)$ and (the runtime limit is not performed)$\Big)$ **do**
7: Branch from the current level ℓ by selecting the ω eligible positions for each node $\eta_{\ell_i} \in B$;
8: Insert all obtained nodes into B_ω;
9: Evaluate the potential of each node belonging to B_ω using GP for completing the path.
10: If a feasible solution is given by GP, then set `feasible` to `true` and **exit**;
11: Replace B by the best ω nodes of B_ω realizing highest densities and, increment the level ℓ.
12: **end while**

Algorithm 11 . A Dichotomous Search Based Heuristic: DSBH

Input. An instance of 3DSPP.

Output. An object $\mathcal{P}$ of dimensions (L_{best}, W, H) and the coordinates of all items of N.

1: **Initialization step**
2: Call an iterative GP on the open strip (∞, W, H) and let $\overline{L}$ be the starting length reached.
3: Set $\underline{L} \leftarrow \frac{4\pi}{3\times W\times H}\sum_{i\in N}(r_i^3)$ and $L^\star = \overline{L}$.
4: Set ω to a predefined minimum value.
5: **Iterative step**
6: **while** (the runtime limit is not performed) **do**
7: **repeat**
8: $L^\star = (\overline{L} + \underline{L})/2$
9: Generate the starting node η^1 with its three sets I_i, $\overline{I}_i$ and P_I^1.
10: Set `feasible` $\longleftarrow$ BS(η^1), where BS is called with $(L^\star, W, H)$
11: **If** (`feasible`=`true`) **then** set $\overline{L} = L^\star$; $\underline{L} = L^\star$ otherwise
12: **until** $(\overline{L} - \underline{L} \geq \alpha)$
13: Set $\underline{L} \leftarrow \frac{4\pi}{3\times W\times H}\sum_{i\in N}(r_i^3)$ and increment ω.
14: **end while**

Finally, the aforementioned process is iterated a certain number of times following the values of ω (line 13) and according to the runtime limit fixed.

4.3 Computational results

In this section we investigate the effectiveness of the width beam search-based heuristic (DSBH) on two sets of benchmark instances: Set1 and Set2.

The first set "Set1" contains six instances (SYS1,...,SYS6) extracted from Stoyan *et al.* [67], where the number of the predefined items varies from 25 to 60. These instances have been already tested using Stoyan *et al.*'s [67], Birgin and Sobral's [8] and Kubach *et al.*'s [46] approaches.

The second set "Set2" contains six instances (KBTG1, KBTG2, KBTG3, KBTG7, KBTG8, and KBTG9) taken from Kubach *et al.* [46]. For each instance, both dimensions W and H of the object are fixed to 10 whereas the number of the predefined items is fixed to 30 (resp. 50) for the first (resp. last) three instances. Moreover, these six instances have been already tested in Kubach *et al.* [46] where they represent the six instances with unequal spheres.

The rest of the experimental part is organized as follows. First, the behavior of the proposed algorithm DSBH is evaluated on the first set of instances Set1 (Section 4.3.1). The results reached by DSBH are thereafter compared to those reached by four heuris-

tics: Stoyan *et al.*'s [67] method-based heuristic (noted SYS), Birgin and Sobral's [8] method-based heuristic and both sequential and parallel heuristics proposed by Kubach *et al.*'s [45, 46] (noted KBTG_s and KBTG_p respectively). Second, for the instances of Set2 (Section 4.3.2), the results provided by DSBH are compared to those reached by Kubach *et al.* [46]. Note also that the proposed algorithm was coded in C++ and tested on an Intel Core 2 Duo (2.53 Ghz and with 4 Gb of RAM) and the runtime limit was fixed to one hour.

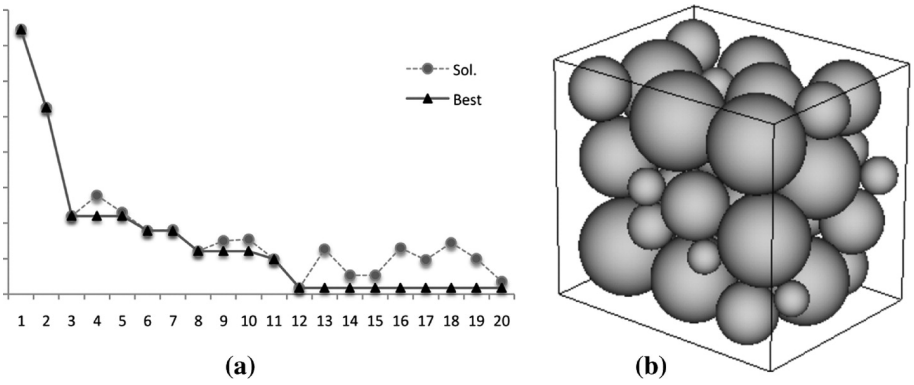


Figure 4.2: Illustration of DSBH's behavior on SYS5 instance: (a) variation of solution's quality and (b) the best solution's structure reached by DSBH.

4.3.1 Performance of DSBH vs three heuristics: Set1

Generally, when using approximate algorithms to solve optimization problems, it is well-known that different parameter settings for the approach lead to results of variable quality. As discussed in Section 4.2.4, DSBH uses two parameters: the beam width ω and the maximum runtime limit to fix. Our computational study was conducted by varying ω in the discrete interval $\{5, 6, 7, \ldots\}$ and the maximum runtime limit was fixed to 3600 seconds (which can be considered as a standard runtime limit considered by algorithms of the literature). Of course, the upper value of ω depends on the limited runtime and the size of the instance. In order to show the effect of these parameters, we discuss the quality of the solutions obtained by DSBH when the beam width ω increases.

First, we do it on an instance extracted from Stoyan *et al.* [67] by varying the value of ω from 1 (the minimum value assigned to ω in Algorithm 11) to the upper value (incrementing ω with one unit) till attaining the fixed runtime limit. Figure 4.2.(a) shows the behavior of DSBH when varying the beam width ω for the instance SYS5. From the curve labeled Sol., one can observe that the value of the length $L^\star$ of the target object $\mathcal{P}$ oscillates but globally its average solution value decreases and converges toward one of the best solutions (generation of a series of local optima). This is the reason why we proposed to reiterate the GP rule in order to get the final solution realizing the best local optimum.

On the other hand, according to the second curve labeled Best, one can observe that the behavior of DSBH provides a series of decreased solution values; in this case, only the best solution was retained for the whole values of ω. For SYS5, the best local optimum reached by DSBH realizes the value closest to 10.94 and its solution configuration is illustrated by Figure 4.2.(b). In fact, the provided solution is equal to 10.9359 (as shown later in the second part of the experimental part) and it dominates both values 11.2170 and 10.9359 reached by the sequential KBTG algorithm (Kubach *et al.* [46]) and its parallel version (Kubach *et al.* [45]) representing the best solution values of the literature. Note that $\omega = 12$ seems to give the best solution value that balances between the quality of the solution and the fixed runtime since for higher values of ω the solution remains unchanged.

Second, for the instances of Set1, Table 4.2 compares the results of DSBH to those reached by four algorithms: SYS (Stoyan *et al.* [67]), BSA (Birgin and Sobral [8]), KBTG$_s$ (Kubach *et al.* [46]) and its parallel version noted KBTG$_s$ (proposed in Kubach *et al.* [45]), where the known solutions of the literature are taken from Kubach *et al.* [45, 46].

				Kubach *et al.*'s [45, 46] heuristics							
	#Inst.			SYS	BSA	KBTG$_s$		KBTG$_p$	DSBH		
Label	n	H	W	$L^\star_{\text{SYS}}$	$L^\star_{\text{BSA}}$	$L^\star_{\text{KBTG}_s}$	$L^\star_{\text{KBTG}_s}$	$L^\star_{\text{KBTG}_p}$	$L^\star$	$L^\star$	ω^*
SYS1	25	5.5	6.9	9.912	9.7942	9.5397	9.2874	9.2656	9.2635	9.2431$^\diamond$	23
SYS2	35	6.5	7.9	9.623	-	9?2608	9.1280	8.9301	9.1585	8.9164 $^\diamond$	21
SYS3	40	5.5	6.9	9.473	9.3090	91794	8.9850	8.7178	8.9085	8.7055$^\diamond$	24
SYS4	45	8.5	9.9	11.086	11.0962	10.9991	10.8760	10.4042	10.7224	10.2357$^\diamond$	23
SYS5	50	8.5	9.9	11.646	11.6211	11.4877	11.3494	10.9865	11.1772	10.9359$^\diamond$	29
SYS6	60	8.5	9.9	12.842	12.7215	12.7110	12.3745	11.8399	12.4149	11.8178 $^\diamond$	28
Average											

Table 4.2: Performance of BSBH versus the four heuristics of the literature on instances of Set1. The symbole " $-$ " (resp. " $\diamond$ ") means that the value for this instance is not available (resp. corresponds to the best solution).

Columns from 1 to 4 show the instance label, problem size n and both height H and width W of $\mathcal{P}$. Column 5 displays the solution value $L^\star_{\text{SYS}}$ reached by STS whereas column 6 displays BSA's solution values (noted $L^\star_{\text{BSA}}$). Columns 7 and 8 report the best solutions (noted $L^\star_{\text{KBTG}}$) provided by KBTG algorithm for both fixed runtimes: 300 and 3600 seconds. Column 8 shows the best solutions reached by the parallel KBTG version (noted $L^\star_{\text{KBTG}_p}$) without fixing the runtime limit. Columns 9 and 10 display the solution (noted $L^\star$) realized by the proposed algorithm DSBH for both fixed runtimes (300 and 3600 seconds, respectively). Finally, column 11 reports the best value of ω for which its best solution for the second runtime limit is performed.

All results of Table 4.2 are summarized in Table 4.3, where it tallies the percentage improvement (when it happens) yielded by DSBH when compared to the results reached by the four considered algorithms. Table 4.3 is composed of two parts: the first part contains the informations about the instance and the second part shows the percentage

	#Inst.			DSBH vs all heuristics (% Improvement)			
Label	n	H	W	%SYS	%BSA	%KBTG$_s$	%KBTG$_p$
SYS1	25	5.5	6.9	7.24	5.96	0.48	0.24
SYS2	35	6.5	7.9	7.92	-	2.37	0.15
SYS3	40	5.5	6.9	8.82	6.93	3.21	0.14
SYS4	45	8.5	9.9	8.31	8.41	6.26	1.65
SYS5	50	8.5	9.9	6.49	6.27	3.78	0.46
SYS6	60	8.5	9.9	8.67	7.65	4.71	0.19
Average				7.91	7.04	3.47	0.47

Table 4.3: Percentage improvements between all tested heuristics: BSBH, SYS, BSA and both KBTG$_s$ and KBTG$_p$ on instances of Set1.

improvement realized by DSBH over the four heuristics tested (noted %SYS, %BSA, %KBTG$_s$ and %KBTG$_p$ according to the heuristics SYS, BSA, KBTG$_s$ and KBTG$_p$, respectively) especially for the second runtime limit for DSBH.
The analysis of the results of both Tables 4.2 and 4.3 follows:

1. First, DSBH outperforms the three algorithms SYS, BSA and KBTG$_s$ whenever the runtime limit was fixed to 300 seconds. Indeed, it is able to reach the best solution for all instances of Set1.

2. Second, DSBH outperforms SYS, BSA and both KBTG$_s$ and KBTG$_p$ algorithms since for the instances of Set1 it is able to provide better results.

3. Third, when comparing DSBH's results to SYS's ones, one can observe that the percentage of the improvement varies from 6.49% (instance SYS5) to 8.82% (instance SYS3). This percentage improvement remains interesting when comparing DSBH's results to those reached by BSA: in this case, such improvement varies between 5.96% (instance SYS1) and 7.65% (instance SYS6). Moreover, it stills positive when comparing DSBH's results to those provided by KBTG algorithm. Indeed, the improvement varies from 0.48% (instance SYS1) to 6.26% (SYS4).

4. Fourth and last, DSBH's results remain better than those reached by the parallel algorithm KBTG$_p$ (we recall that the parallel algorithm ran without runtime limit). Indeed, all the solution values for the instances of Set1 are improved. In this case, the percentage of improvement varies from 0.14% (instance SYS3) to 1.65% (instance SYS4), which remains very interesting for a sequential algorithm.

Figure 4.3 shows the behavior of DSBH on the instances of Set1 where each curve represents the variation of the improvements realized according to the algorithm SYS,

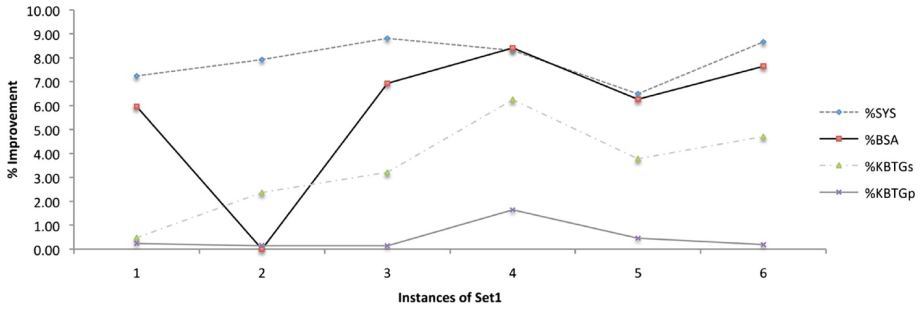


Figure 4.3: Variation of the percentage improvement realized by DSBH when compared to the results of the four algorithms (SYS, BSA and both KBTG$_s$ and KBTG$_p$) on the instances of Set1.

BSA and both KBTG$_s$ and KBTG$_p$, respectively. Figure 4.4.(a) illustrates GP's configuration (when applied as a heuristic for providing a starting solution for DSBG) and Figure 4.4.(b) displays DSBH's configuration. From the starting configuration (cf., Figure 4.4.(a)), one ca observe the failure of GP when used as a heuristic: in this case, all items are concentrated on the bottom-left-depth position and all positions located along the length of $\mathcal{P}$ to the top are omitted. However, the configuration performed by DSBH shows the correction made when GP's process is repeated.

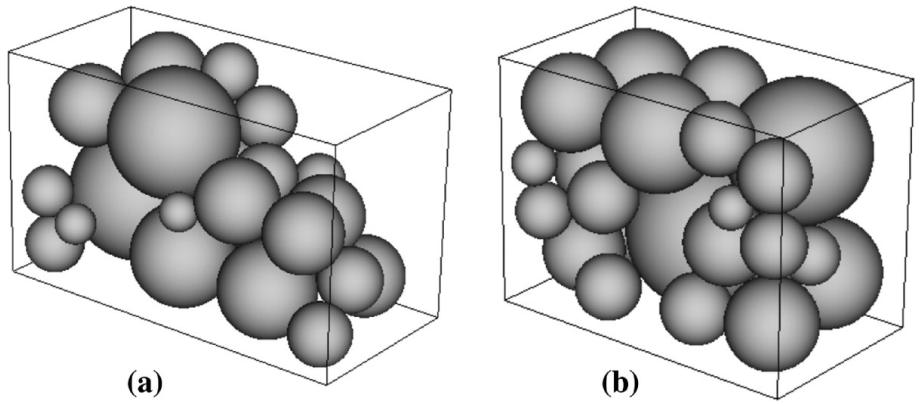


Figure 4.4: Illustration of DSBH's behavior on SYS1 instance (Set1: (a) the starting solution with a length equal to 11.5051 and (b) the best solution's structure and its length (9.2431) reached by DSBH.

4.3.2 Performance of DSBH versus KBTG heuristic: Set2

This section compares the results reached by DSBH to those reached by KBTG$_s$ (note that, for this type of instances, KBTG$_s$ realizes the best solution values of the literature). This comparison is performed on the instances of the second group Set2 extracted from

Kubach *et al.* [46]. For this type of instances, instead of determining the minimum length $L^\star$ of the target container $\mathcal{P}$, Kubach *et al.* [46] computed the density of all packed items in the final object $\mathcal{P}$. Therefore, in addition to the density, the best length $L^\star$ of the final object $\mathcal{P}$ corresponding to that density is reported in the same table.

#Inst.					Dichotomous Search-Based Heuristic			
Label	n	H	W	$d^\star_{KBTG_s}$	$L^\star$	$d^\star$	ω^*	$\%KBTG_s$
KBTG1	30	10	10	54.096	10.9031	56.0092$^\diamond$	18	3.42
KBTG2	30	10	10	30.071$^\diamond$	1.9900	30.071$^\diamond$	23	0.00
KBTG3	30	10	10	51.387	18.2415	53.6243$^\diamond$	21	4.17
KBTG7	50	10	10	55.372	13.0997	57.5662 $^\diamond$	17	3.81
KBTG8	50	10	10	45.060	2.5825	47.004$^\diamond$	15	4.14
KBTG9	50	10	10	52.732	27.8033	55.3203$^\diamond$	13	4.68
Average				48.120		*49.932*		3.63

Table 4.4: Performance of BSBH versus KBTG$_s$ on instances of Set2.

The results realized by the two tested methods (DSBH and KBTG$_s$, respectively) are reported in Table 4.4. Columns 1 to 4 display all the characteristics related to the instance. Column 5 reports the solution value (expressed in term of density) realized by KBTG$_s$'s algorithm (extracted from Kubach *et al.* [45, 46]). The next three columns report the best length $L^\star$ reached by DSBH, its corresponding density and the best value of ω for which the best solution is reached. Finally, the last column displays the percentage improvement realized by DSBH according to the solution values reached by KBTG$_s$. The analysis of the results of Table 4.4 follows.

1. DSBH remains competitive since it improves most solutions reached by KBTG$_s$. Indeed, it is able to improve five out of six best solutions while it matches the other solution (instance KBTG2) when compared to the results reached by KBTG$_s$.

2. For the improved solutions, DSBH realizes an improvement varying from 3.42% (instance KBTG9) to 4.68% (instance KBTG8). Globally, the average improvement over all instances is equal to 3.63%.

3. In term of density, DSBH realizes a density of 49.932% which is much better than those realized by KBTG$_s$ (48.120%), as illustrated in Figure 4.5.

Finally, Figure 4.6.(a) illustrates the starting solution reached by iterating GP till packing all the items whereas Figure 4.6.(b) shows the final solution when DSBH is applied. We can observe that the starting configuration leaves also an important unoccupied area along the length of the object $\mathcal{P}$ to the depth. However, the configuration leads a better packing when GP rule is repeated on different eligible positions of the search space.

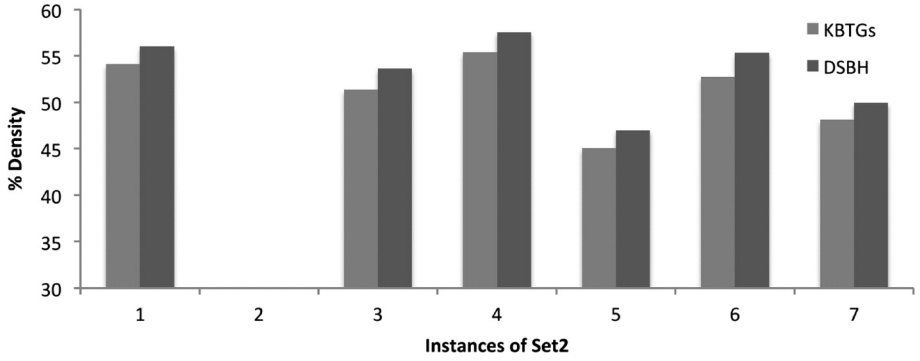


Figure 4.5: Variation of the percentage of the density realized by both DSBH and KBTG$_s$ on the instances of Set2.

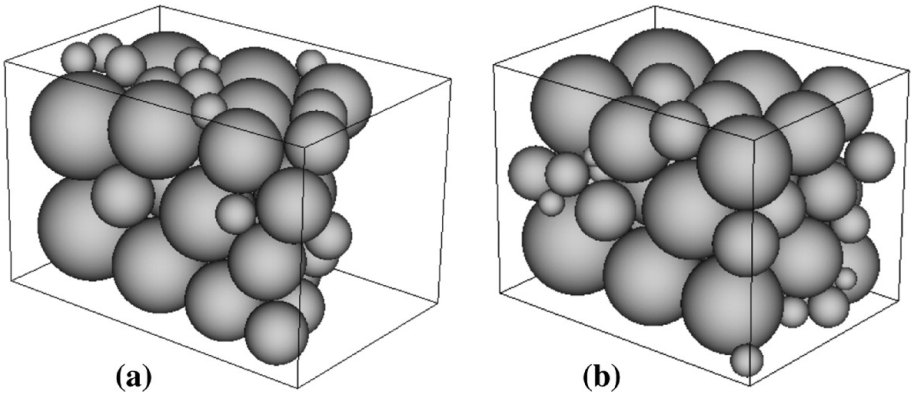


Figure 4.6: Illustration of DSBH's behavior on KBTG7 instance (Set2): (a) the starting solution with a length equal to 15.3817 and (b) the best solution's structure and its length (13.0997) reached by DSBH.

4.4 Conclusion

In this paper the three-dimensional sphere packing problem is solved by using a dichotomous search-based heuristic. The proposed method is based upon three complementary phases: (i) a greedy selection phase which tries to select many eligible positions to iteratively packing all predefined items into the target object, (ii) a width beam search phase that serves to explore some promising paths, and (iii) a dichotomous search that serves to diversify the search space. The first two phases iterated until reaching the final objet; that is, the container with the smallest length containing all the items. The performance of the proposed method is evaluated on benchmark instances of the literature where the provided results are compared to those reached by some recent methods of the literature. The proposed method remains competitive and succeeded in yielding new solutions on many instances.

Chapter 5

A global dichotomous search-based heuristic for the three-dimensional sphere packing problem

Contents

In this chapter, the three-dimensional sphere packing problem is tackled by using a global dichotomous search-based heuristic. An instance of the problem is characterized by a set N of n spheres and an open container with unlimited length. The goal of the problem is to determine the minimum length of the container that contains all spheres without overlapping. We propose to optimize the length of the large container by applying a truncated tree-search that combines a hill-climbing strategy, a hybrid operator that combines both priority and total-cost operators and, a dichotomous interval search in order to diversify the search space. Further, in order to enhance the quality of solutions of internal nodes, a local dichotomous search is applied almost of using a descent method.

5.1 Introduction

Herein, we study the problem of packing spheres (namely items) in a container (namely parallelepiped) with unlimited length. The problem is known as the *Three-Dimensional Sphere Packing Problem* (3DSPP), where an instance of the problem is defined by a set N of n unequal items (each item $i \in N = \{1, \ldots, n\}$ is represented by its radius r_i) and a container $\mathcal{P}$ of fixed width W and height H and, unlimited length (for the rest of the paper, the length representing the variation of the length of $\mathcal{P}$ is denoted by L). The goal of 3DSPP is to minimize the length L of the container $\mathcal{P}$ such that all items of N are positioned in the current container, without overlapping. Formally, 3DSPP can be stated as follows:

$$\text{Minimize} \quad L \tag{5.1}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \geq (r_i + r_j), \qquad \forall (i,j) \in N^2,\ i < j \tag{5.2}$$

$$r_i \leq x_i \leq L - r_i,\ \forall i \in N \tag{5.3}$$

$$r_i \leq y_i \leq H - r_i, \forall i \in N \tag{5.4}$$

$$r_i \leq z_i \leq W - r_i, \forall i \in N \tag{5.5}$$

$$\underline{L} \leq L \leq \overline{L} \tag{5.6}$$

$$(x_i, y_i, z_i) \in \mathbb{R}^3_+,\ \ \forall i \in N. \tag{5.7}$$

where (x_i, y_i, z_i), $i \in N$, denote the coordinates of the center of sphere i, and $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ is the Euclidean distance separating the centers of spheres i and j, $1 \leq i \leq j \leq n$, $i \neq j$. This model has a linear objective function (Equation (5.1)) but quadratic (Equation (5.2)) and linear (Equations from (5.3) to (5.6)) constraints. Indeed, Equation (5.2) ensures the non-overlap constraint of any pair of distinct items (i, j) of $N \times N$ and Equations (5.3) to (5.5) ensure that all items of N belong to the current container $\mathcal{P}$ of dimensions (L, W, H). Finally, Equations (5.7) ensure that all items are placed in the container $\mathcal{P}$.

We propose a global dichotomous search-based heuristic for the three-dimensional sphere packing problem. Such an approach mimics the branch-and-bound procedure (cf., Moslehi *et al.* [58], Solimanpur *et al.* [65] and Ait Zai *et al.* [1] for more recent works addressing sequential and parallel tree-search techniques for solving different variants of optimization problems), where from a selected node, all branches are created following a hybrid operator that combines both priority and total-cost operators. The hybrid operator can be viewed as a two-stage procedure: (i) the first stage applies a greedy solution procedure for obtaining a probable feasible solution (according to the subset of items already packed) whereas although it is necessary to be accurate to provide each node a closest value to its objective function, (ii) the second stage proposes to simulate a lower bound for the rest of non packed items. Finally, almost of bounding the search (of the

smallest length of the final contained container $\mathcal{P}$) with a global interval, we propose to introduce a local dichotomous search which can be applied directly to each eligible internal node. In this case, using such a procedure may serve to achieve a series of lengths (local minima), thereby to converge quickly to the best global optimum.

In fact, the proposed method is based on combining three main features:

(i) Creating a partial solution: a subset of paths can be reached by applying a quick basic greedy constructive procedure. Indeed, each path represents a subset of items already positioned into the current container. The extremity of each path corresponds to an eligible internal node of the created tree. In this case, the current subset of retained nodes are selected by applying a hill-climbing strategy that is based upon a hybrid evaluation operator.

(ii) Using a local dichotomous procedure: for each created path, a complementary solution can be obtained by applying an extended version of a quick basic greedy procedure. Indeed, in order to accelerate the convergence of the method, we propose to bound the search process with an interval search $[\underline{L}, \overline{L}]$, where $\overline{L}$ denotes the best upper bound (feasible solution) reached up to now and $\underline{L}$ a lower bound that can be estimated for the rest of the subproblem (representing the items not yet packed).

(iii) Applying an iterative search: restarting the search implies the generation of a new starting node and so, a diversification strategy is applied in order to explore different search spaces.

The remainder of the chapter is organized as follows. Section 5.2.1 discusses the problem representation and Section 5.2.2 shows how to compute the subset of eligible positions when a selected item is chosen. Section 5.2.3 exposes a process that is based upon a tree search combined with both hill-climbing and dichotomous search strategies. Given a current container, the tree search is employed for generating eligible nodes to develop, the hill-climbing mimics the global search where all eligible positions are evaluated following an evaluation operator and, the dichotomous search is applied to each retained node. Such a combination is iterated until obtaining the best length associated to the final container. Such a process is embedded into a global dichotomous search for diversifying the solutions. Section 5.3 evaluates the performance of the proposed algorithm where its obtained results are compared to those reached by recent algorithms available in the literature. Finally, Section 5.4 concludes by summarizing the contribution of the paper and proposes some eventual perspectives.

5.2 A global dichotomous search-based heuristic for 3DSPP

This section starts by exposing the problem representation (Section 5.2.1). The branches, characterizing a series of nodes of the developed tree, are generated following a basic

procedure as described in Section 5.2.2. Finally, Section 5.2.3 exposes the principle of the proposed algorithm and its main steps. Note that in order to make the paper self-contained, we repeat some parts already discussed in Hifi and Yousef [37, 38, 36].

5.2.1 Representation of an instance of 3DSPP

Generating eligible nodes for the developed tree needs the representation of the objet and each item. The following representation is used:

- Assume that the bottom-left-depth corner of the current container $\mathcal{P}$ is positioned at $(0, 0, 0)$, where $\mathcal{P}$ is represented by a set of faces, namely $\mathbb{F} = \{\text{left}, \text{top}, \text{right}, \text{bottom}, \text{depth}, \text{front}\}$. Then, $\mathcal{P}$ is represented in the Euclidian space, as illustrated in Figure 5.1.
- Each item $i \in N$ is centered at the position (x_i, y_i, z_i).
- A pair (i, j) of N are represented by their distance $\delta_{i,j}$:

$$\delta_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} - (r_i + r_j), \quad \forall (i, j) \in N^2, \tag{5.8}$$

where both i and j can be positioned when $\delta_{i,j} \geq 0$.

Table 5.1: The distance between both item i and a face f

f	$\delta_{i,f} \mid i \in N,\ f \in \mathbb{F}$
left	$x_i - r_i$
bottom	$y_i - r_i$
depth	$z_i - r_i$
right	$L - x_i - r_i$
top	$H - y_i - r_i$
front	$W - z_i - r_i$

Furthermore, positioning $i \in N$ into the current container $\mathcal{P}$ needs to define a distance between i and the faces of $\mathbb{F}$. Indeed, for all faces of $\mathcal{P}$, Table 5.1 shows each corresponding distance to be satisfied if $i \in$ is assigned to an eligible position of $\mathcal{P}$.

5.2.2 (Sub)paths: partial and complete solutions

Generating eligible paths, corresponding to the final nodes of the developed tree, is equivalent to generate a subset of positions in the current container $\mathcal{P}$. It can be done by applying the so-called *Basic Greedy Procedure* (namely BGP) (cf. Hifi and Yousef [38, 36]). Indeed, BGP tries to find the minimum distance between already packed items and the current selected item to pack in $\mathcal{P}$. However, in order to curtail the cardinality of the set of positions, only a subset of eligible positions are considered (according to the positions defined in Table 5.1)

We recall that all generated positions are used for creating eligible branches (sub-paths) according to a selected node. BGP is then used as an operator for evaluating the potential of a node (position) to investigate. Therefore, BGP is used either for providing a complete feasible solution or for estimating the gap between the node's upper bound and the best length reached up to now.

5.2.2.1 BGP and eligible positions

Let $i \in N$ be the current item to be packed into $\mathcal{P}$ and P_{I_i} be the subset of its eligible positions. The first item of N is packed at the position (r_1, r_1, r_1) and for $i \in N,\ i \geq 2$, the following notations are considered:

- I_i: the set of items of N already packed in the current container $\mathcal{P}$.
- $\overline{I_i}$: the set of items of N which are not yet packed in the current container $\mathcal{P}$.
- P_{I_i}: the set of distinct eligible positions for the next item i to pack given the set of packed items I_i.

It follows that an eligible position $p_{i+1} \in P_{I_i}$ (for the item i) is determined according to three elements, namely $e_1,\ e_2$ and e_3. Of course, an element is either an item of N already positioned (representing I_i) or one of the six faces belonging to $\mathbb{F}$. Finally, assume that $\mathcal{T}_{p_{i+1}}$ denotes the set of nodes according to $e_1,\ e_2$ and e_3. A step of BGP, for $i \geq 2,\ i \in N$, proceeds as follows: the distance $\Delta(p^k_{i+1})$ characterizing the $(i+1)$-th item to pack, when positioned at the eligible position $p^k_{i+1} \in P_{I_i}$, realizes what follows:

$$\Delta(p^k_{i+1}) = \min_{j \in I_i \cup \mathbb{F} \setminus \mathcal{T}_{p^k_{i+1}}} \delta_{(i+1,j)}. \tag{5.9}$$

Hence, BGP starts by positioning the first item $i = 1$ at the bottom-left-depth position, i.e., at the position (r_1, r_1, r_1), while the remaining $n-1$ items are successively positioned according to $\Delta(.)$ (cf., Equation (5.9)). Note that if different eligible positions have the same distance, then one can choose either (i) the first item according to the order initially fixed or (ii) the item that realizes the smallest distance to the left side of $\mathcal{P}$.

Finally, one can use BGP as a greedy procedure for searching a feasible solution to 3DSPP. Indeed, it can be realized by setting the starting length of the target container $\mathcal{P}$ to ∞ (i.e., $L_{\mathcal{P}} = \infty$) and by applying successively BGP (by excluding the "right" face from $\mathbb{F}$) on the rest of the non packed items (BGP stops with a feasible solution with length L_U). The provided length L_U can also be considered as an upper bound for starting the proposed algorithm.

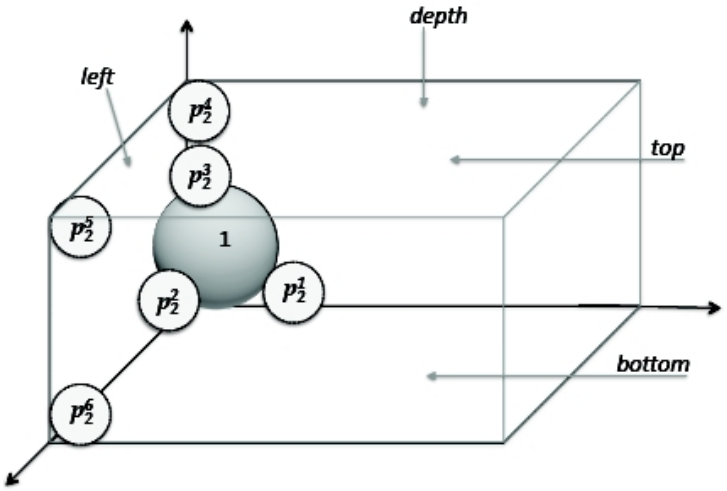


Figure 5.1: Illustration of the current container $\mathcal{P}$ and the mechanism used for generating eligible positions (nodes).

Example

Assume that the first item is already positioned in the container $\mathcal{P}$ at position (r_1, r_1, r_1). Figure 5.1 shows six eligible positions that emerge for the next item 2 to pack. By using the representation discussed in Section 5.2.1, $I_1 = \{1\}$ and $P_{I_2} = \{p_2^j,\ j = 1, \ldots, 6\}$:

- First, the position p_2^1 touches the item 1 and both faces "depth" and "bottom" of $\mathcal{P}$.
- Second, the position p_2^2 is obtained by using the item 1 and both faces "left" and "bottom" of $\mathcal{P}$.
- Third, the position p_2^3 is computed by using the item 1 and both faces "left" and "depth" of $\mathcal{P}$.
- Fourth and last, all other positions p_2^4, p_2^5 and p_2^6touch three faces of $\mathcal{P}$.

It follows that $\mathcal{T}_{p_2^1} = \{1, \text{depth}, \text{bottom}\}$, $\mathcal{T}_{p_2^2} = \{1, \text{left}, \text{bottom}\}$, $\mathcal{T}_{p_2^3} = \{1, \text{left}, \text{depth}\}$, $\mathcal{T}_{p_2^4} = \{\text{top}, \text{left}, \text{depth}\}$, $\mathcal{T}_{p_2^5} = \{\text{top}, \text{left}, \text{front}\}$ and $\mathcal{T}_{p_2^6} = \{\text{bottom}, \text{left}, \text{front}\}$.

Finally, as illustrated in Figure 5.1, the item 1 can be packed at one of the six positions $p_2^j,\ j = 1, \ldots, 6$, and if the smallest distance to the left side of $\mathcal{P}$ is favored, then five positions remains favorable.

5.2.2.2 Enhancing BGP

Now suppose that instead of packing the n items in $(\overline{L}, W, H)$, we propose to perform the search on a series of containers of dimensions (L_k, W, H), with $\underline{L} \leq L_k \leq \overline{L}$. In this case,

one can use a starting interval $[\underline{L}, \overline{L}]$, where $\underline{L}$ denotes a lower bound for the 3DSPP and $\overline{L}$ its upper bound and so, for each target container (L_k, W, H), BGP attempts to pack all the n items into (L_k, W, H), where $\underline{L} \leq L_k \leq \overline{L}$. As a result, a series of containers $\mathcal{P}_k,\ k = 1, 2, \ldots$ are explored using a binary search.

Algorithm 12 . Enhancing BGP (EBGP)

Input. A node η and the best length $L^\star$.

Output. A length $L^\star$ and the coordinates of all items of N.

1: **Initialization step**
2: Set $\underline{L} \leftarrow \frac{4\pi}{3\times W\times H}\sum_{i\in N}(r_i^3)$ and $\overline{L} = L^\star$.
3: **Iterative step**
4: **repeat**
5: $L' = (\overline{L} + \underline{L})/2$
6: Set `feasible` = BGP(η)
7: **if** `feasible then`
8: `set` $\overline{L} = L'$, $L^\star = \min\left\{L^\star, L'\right\}$; $\underline{L} = L'$ `otherwise`
9: `end if`
10: **until** $(\overline{L} - \underline{L} \geq \epsilon)$ or (the runtime limit is performed)

Algorithm 12 describes the framework of the extended version of the basic greedy procedure.

5.2.3 Using a tree search

Generally, solving a combinatorial optimization problem with a tree search requires (i) to define the nodes with their characteristics and, (ii) to establish the branching mechanism out of the nodes which induces the successors.

5.2.3.1 Positions and eligible nodes

The *valid nodes* represent a subset of *eligible positions* of the tree search. Each selected node η_i to develop is characterized by a pair of subsets:

- I_i : the first subset that contains all items assigned to the current container $\mathcal{P}$.
- $\overline{I}_i$: the complementary subset that includes the unassigned items.

A *partial solution* depends on the positioned nodes of I_i whereas a *complementary solution* can be reached according to all non-positioned nodes belong to $\overline{I}_i$.

5.2.3.2 Branching

Branching out of a selected node η_i is equivalent to create at most $|P_{I_i}|$ branches. Each induced node is related to positioning all items of I_i and assigning a new position for the

current item i. Moreover, the assessment of the potential of each valid node is considered as a critical step for enumerative methods. The potential of each node may be evaluated by its lower and upper bounds. For the 3DSPP, on the one hand, an upper bound can be computed by using either a greedy procedure or a more sophisticate heuristic. On the other hand, a lower bound can be computed by using a simple and quick approximate solution value. Of course an approximate value may be used because it is not easy to provide each node η_i a potential value that is close to the objective function value of any leaf node emerging from η_i. So, the evaluation operators are introduced for estimating the assessment of the potential of each node.

Generally two operators may be distinguished when an enumerative method is applied:

- A *priority operator*: it can be viewed as a local operator that uses a greedy heuristic to build a complete solution.
- A *total-cost operator*: it can be viewed as a global operator that is able to estimate the potential of the current node η by estimating the objective value of a complete solution.

Algorithm 13 . A standard Tree Search-Based Algorithm (TSBA)

Input. An instance of a minimization problem.

Output. An optimal solution η^* with objective value z^*.

Initialization Step.
Set $Open = \{\eta_0\}$, where $Open$ is the set of nodes to be investigated and η_0 is the root node.
If an initial feasible solution $\eta^\star$ is available, set $z^\star$ to its objective function value; otherwise, set $z^\star = +\infty$.
Iterative Step.
while ($Open \neq \emptyset$) **do**
    Choose a node $\eta \in Open$; branch out η; remove η from $Open$ and insert the created nodes into B_η.
    if (a node γ of B_η is a leaf) **then**
        compute its objective function value z_γ;
        if $z_\gamma < z^\star$ **then**
            update $z^\star$ and the incumbent solution $\eta^\star$;
        end if
        remove γ from B_η.
    end if
    Assess the potential of each node of B_γ using an evaluation operator.
    Insert all nodes of B_γ into $Open$.
end while

The main steps of the standard tree search, for a minimization problem, may be described by the steps of Algorithm 13. Each node η is characterized by a partial (feasible) solution and a set $Open$ of current nodes that is initialized at the root node, namely η_0. Each $\eta \in Open$ creates a set of offspring nodes, and stores them into a temporary list B_η. In the case where the node γ of B_η is a leaf, then its objective function value z_γ is computed and compared to $z^\star$; that is, the best objective value obtained up to now. If z_γ is less than $z^\star$, then the incumbent solution is replaced by the leaf node and $z^\star$ is updated

as follows: $z^\star = z_\gamma$. In this case, γ is removed from B_γ. Finally, all nodes belonging to the set B_η are moved to *Open* for further branchings. This process is iterated until $Open = \emptyset$; in this case, no further branching is possible and so, the algorithm stops.

5.2.4 Using TSBA for solving 3DSPP

Algorithm 14 . Global Search-Based Heuristic for 3DSPP (GSBH)

Input. A starting length $L^\star$.

Output. An improved length $L^\star$ and the coordinates of all packed items of N.

1: **Initialization Step.**
2: Let ω and ϵ be two predefined values.
3: Set $Open = B_0$, where B_0 denotes the starting eligible nodes according to the first packed item $i = 1$.
4: Set the variable `feasible` to `false` /* no feasible solution at hand */
5: **Iterative Step.**
6: **while** $\Big((Open \neq \emptyset)$ and (the runtime limit is not performed)$\Big)$ **do**
7: Choose η from *Open*;
8: Let $B_\eta = \{\gamma_1, \ldots, \gamma_{|P_{I_\eta}|}\}$ be the successors of η.
9: Evaluate the potential of each node γ belonging to B_η by computing $g(\gamma)$ and $h'(\gamma)$.
10: For each $\gamma \in B_\eta$ apply EBGP$(\gamma, L^\star)$ and update $L^\star$ if necessary with the incumbent solution (in this case, set `feasible` to `true`).
11: Filter B_η by keeping the ω best nodes realizing the smallest values of $L^\star/\big(g(\gamma)+h'(\gamma)\big)$.
12: Transfer all the nodes of B_η to *Open* and reduce B_η to empty.
13: **end while**

5.2.4.1 A global truncated tree-search-based heuristic

Algorithm 14 describes an adaptation of the truncated tree-search-based heuristic (called beam search in Hifi and Saadi [34] and Yavuza [76]). We recall that a node corresponds to a partial solution and the set *Open* of current nodes contains initially the starting nodes of the root node B_0 whereas B_ω containing the offspring nodes is initialized to the empty set.

On the one hand, a selected node η taken from *Open* (step 7), whose evaluation is z_η, creates a subset of nodes $B_\eta = \{\gamma_1, \ldots, \gamma_{|P_{I_\eta}|}\}$, where each resulting node is evaluated according to its global-cost operator; that is,

$$z_\eta = g(\eta) + h(\eta).$$

On the other hand, because $|P_{I_\eta}|$ may be large, then only a subset of nodes is chosen for further branching. Indeed (lines 8-12), if a node γ of B_η packs at most $n-1$ items, then

it remains in B_η whenever $z'(\gamma) < z^\star$, where

$$z'(\gamma) = g(\eta) + h'(\eta) \tag{5.10}$$

with $h'(\eta) = (1+\epsilon)h(\eta)$ and ϵ is considered as a small predefined value that is used for making a correction on the complementary lower bound $h(\eta)$.

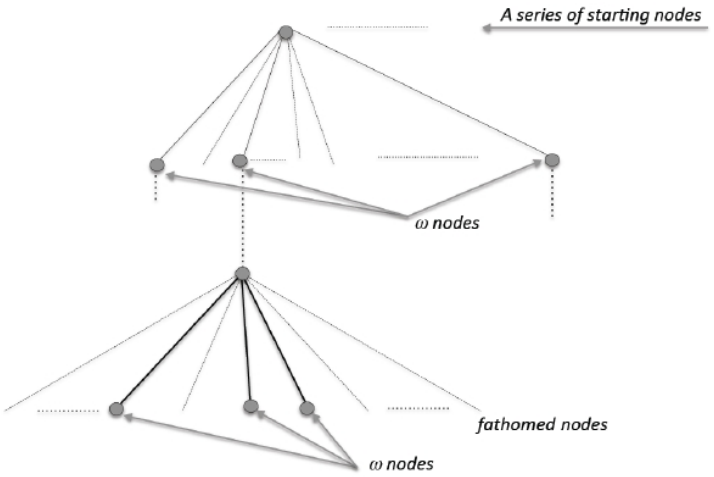


Figure 5.2: Selecting favorable nodes and fathoming unsearched ones.

Whenever Equation (5.10) is not satisfied, then γ is removed from B_η. Further, since we try to intensify the search that permits to improve the quality of the solution, we apply EBGP on all retuned nodes (line 10). Then, $L^\star$ is updated whenever EBGP realizes a better length; in this case, its corresponding incumbent solution is also updated. The rest of the nodes belonging to B_η (line 11) are reordered in nondecreasing order of their estimated lower bounds $z'(\gamma)$ and only the best ω nodes are selected and transferred to *Open* for further branchings. This process is iterated until no further branching is possible, i.e., until $Open = \emptyset$, or when the fixed runtime limit is performed.

Note also that, at lines 10 and 11, if a node γ of B_η is a leaf (i.e, no further branching is possible out of γ), then its objective function value z_γ is computed and compared to the best solution value $z^\star$ obtained up to now. If $z_\gamma < z^\star$, then the incumbent solution is set to a leaf node; $z^\star$ is then updated: $z^\star = z_\gamma$; and γ is removed from B_η.

Figure 5.2 illustrates a developed tree provided after a certain number of branchings. First, the root node contains the list previously denoted B_0, where a node η is selected for further branchings. Second, only ω nodes were selected (cf., Figure 5.2) and the other nodes are discarded. Third and last, at an internal level of the tree, one can observe that

the search process may select a node following either the best-first search or depth-first search strategies. The first selected node which smallest lower bound is selected whenever the best-first search strategy is considered whereas the node which placed more items (and realizing the smallest partial solution value) is selected otherwise.

5.2.4.2 Bounding the search process

It was already mentioned in Section 5.2.4 (see Step 10 of Algorithm 14) that GSBH uses diversification search in order to improve the quality of solutions reached. Herein, the principle of such a procedure is explained.

Let $(\overline{L}, W, H)$ be the current container $\mathcal{P}$ and η be the node chosen for branching. The length $\overline{L}$ denotes the best length reached by DSBH at a certain time. Now suppose that instead of packing the n items in $(\overline{L}, W, H)$, we propose to perform the search on a series of containers of dimensions (L_k, W, H), with $\underline{L} \leq L_k \leq \overline{L}$. In this case, one can use a starting interval $[\underline{L}, \overline{L}]$, where $\underline{L}$ denotes a lower bound for the 3DSPP and $\overline{L}$ its upper bound and so, for each target container (L_k, W, H), BGP attempts to pack all the n items into (L_k, W, H), where $\underline{L} \leq L_k \leq \overline{L}$.

Algorithm 15 summarizes the principle of such a process which can be viewed as a diversification procedure. Indeed, it begins (line 2) by defining the starting interval $[\underline{L}, \overline{L}]$, where $\overline{L}$ is setting equal to the best length $L^\star$ reached so far. The internal loop (lines 5-11) tries to pack all items in the target container by using EBGP. If a feasible solution is obtained, then the incumbent solution is stored with its best length ($L^\star = \overline{L}$), where a new reduced interval is considered. The process is iterated until the gap between both lower and upper bounds (of the current interval) becomes closest to a certain tolerance, namely α (that is fixed to 0.1 in our experimental study). Finally, the aforementioned process can also be stopped when the fixed runtime limit is exceeded.

5.3 Computational results

The performance of the proposed method (noted GDSBH) was evaluated on three sets of instances (Set1, Set2 and Set3) available in the literature. The first set Set1 has been used as benchmarks in Stoyan *et al.* [67], Birgin and Sobral [8] and Kubach *et al.* [46]. It contains six instances (noted from SYS1 to SYS6) taken from Stoyan *et al.* [67], where the number of items varies from 25 to 60. The second set Set2 has been used as benchmarks in Kubach *et al.* [46], where they represent instances containing unequal spheres. It contains six instances (noted KBTG1, KBTG2, KBTG3, KBTG7, KBTG8, and KBTG9) which are taken from Kubach *et al.* [46]. For each instance, the dimensions W and H of the container are fixed to 10 and the number of items is setting equal to 30 (resp. 50) for the first (resp. last) three instances. Finally, the last set Set3 contains 22 large-scale instances that are obtained by combining the six standard instances of Stoyan *et al.* [67]. All tested

Algorithm 15 . Global Dichotomous Search-Based Heuristic (GDSBH)

Input. A starting length $L^\star$.

Output. The best length $L^\star$ and the coordinates of all items of N.
/* with a new length $L^\star$ and the coordinates of all items of N when `feasible = true` */

Initialization step
Set $\underline{L} \leftarrow \frac{4\pi}{3\times W\times H}\sum_{i\in N}(r_i^3)$ and $\overline{L} = L^\star$.
Iterative step
while (the runtime limit is not performed) **do**
    repeat
        $L' = (\overline{L} + \underline{L})/2$
        Set `feasible` = GSBH(η, $L^\star$)
        if `feasible then`
            `set` $\overline{L} = L'$, $L^\star = L'$ `and return` $L^\star$; $\underline{L} = L'$ `otherwise`
        `end if`
    until $(\overline{L} - \underline{L} \geq \alpha)$
end while

algorithms were coded in C++ and tested on an Intel Core 2 Duo (2.53 Ghz and with 4 Gb of RAM) and, the runtime limit was fixed to one hour (or two hours when large-scale instances are tested).

The rest of this part is organized as follows. First, GDSBH's behavior is evaluated on Set1 (Section 5.3.3). GDSBHs' results are thereafter compared to those reached by the best five heuristics available in the literature: Stoyan *et al.*'s [67] heuristic (noted SYS), Birgin and Sobral's [8] algorithm (noted BSA), both sequential and parallel heuristics proposed by Kubach *et al.* [45, 46] (noted KBTG$_s$ and KBTG$_p$ respectively) and Hifi and Yousef's [37, 36, 38] algorithm (noted HY) that uses both standard width-beam search and its improved version. Second, for the instances of Set2 (Section 5.3.4), the results provided by GDSBH are compared to those reached by Kubach *et al.* [46] and Hifi and Yousef [37, 36, 38]. Third and last, the instances of Set3 are tested by comparing the results reached by both HY (Hifi and Yousef [38]) and the proposed GDSBH.

5.3.1 Behavior of GDSBH versus HY on SYS5

In order to show the effect of the dichotomous search, used as an intensification strategy, we discuss the quality of the solutions obtained by both BGP and EBGP; its extended version. We do it on the instance SYS5 considered by Stoyan *et al.* [67]. On the one hand, Figure 5.3.(a) shows the solution's structure reached by CGP whose value is equal to 12.7396 for the instance SYS5. On the other hand, one can observe that (see Figure 5.3.(b)) the value of the length obtained by EBGP ($L_{EBGP} = 11.2671$) is better than that realized by BGP.

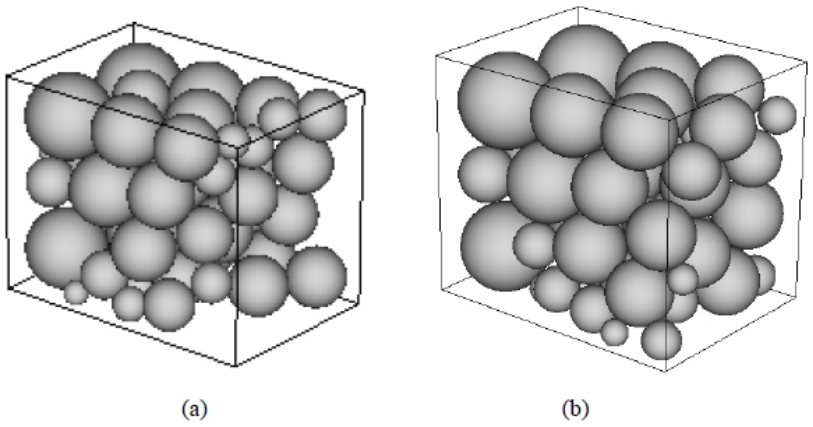


Figure 5.3: Behavior of BGP with the dichotomous search on the instance SYS5: (a) the starting solution (with $L_{BGP} = 12.7396$) reached by BGP and (b) the solution's structure (with $L_{EBGP} = 11.2671$) reached by the extended version of BGP (EBGP).

Hence, according to both solutions, one can expect the good behavior of the proposed method which applies EBGP as an intensification strategy instead as a simple descent method.

5.3.2 Parameter settings

In the preliminary results, three parameters should be taken into account by GDSBH (cf., Algorithm 14): the value associated to ϵ, the size of the global list *Open* that maintains the diversity and the best solutions of the problem and, the maximum runtime fixed to one hour (which can be considered as a standard runtime limit considered by algorithms of the literature). Of course, we recall that GDSBH stops before matching the fixed runtime limit of one hour

	Variation of the parameter ϵ						
Size	0.10	0.15	0.20	0.25	0.30	Av. Min	Av. Max
1000	9.9886	9.9178	9.8514	9.8985	9.9168	9.8514	9.9886
2000	9.9319	9.9055	**9.8357**	9.8550	9.9012	**9.8357**	9.9319
3000	9.9607	9.9316	9.8493	9.9569	9.9727	9.8493	9.9727

Table 5.2: GDSBHs average solution values on instances of Set1.

Table 5.2 shows the average values corresponding to all solutions obtained by GDSBH over instances of Set1 by fixing the same average runtime (3600 seconds, on average). The first column reports the variation of the size of the global list *Open* (noted $\rho = |Open|$) that varies in the discrete interval $\{1000;\ 2000;\ 3000\}$. Columns from 2 to 6 show the average results when varying ϵ in the interval $\{10\%; 15\%; 20\%; 25\%; 30\%\}$ and the last two columns give both smallest and greatest average values over all treated instances with the same values for ϵ.

From Table 5.2, one can observe that, for the same runtime limit, globally the average quality of the solutions obtained is better when fixing (ρ, ϵ) to $(2000, 20\%)$ (the cell corresponding to line 2 and column 4 of Table 5.2). Indeed, on the one hand, the average of solutions decreases until the value of 20% for ϵ. In this case, one can remark that the quality of the solutions is always realized for this value, i.e., $\epsilon = 20\%$. On the other hand, increasing the value of the parameter ρ induces the degradation of the quality of the solutions (the same phenomenon is observed whenever ρ is fixed to the smallest value 1000). Then, for the rest of the experimental part, we used the couple $(\rho, \epsilon) = (2000, 20\%)$ which provides the best average solution values for the first set of instances.

5.3.3 Behavior of GDSBH versus available heuristics on Set1

In this part, the behavior of GDSBH is evaluated on instances of Set1. Table 5.3 shows the results obtained by all tested heuristics: GDSBH, SYS (Stoyan *et al.* [67]), BSA (Birgin and Sobral [8]), KBTG$_s$ (Kubach *et al.* [46]), its parallel version KBTG$_p$ (proposed in Kubach *et al.* [45]) and HY denoting the three versions of the width-beam search proposed in (Hifi and Yousef [37, 36, 38]).

#Inst.				SYS	BSA	KBTG$_s$	KBTG$_p$	HY	GDSBH
Label	n	H	W	$L^\star_{\text{SYS}}$	$L^\star_{\text{BSA}}$	$L^\star_{\text{KBTG}_s}$	$L^\star_{\text{KBTG}_p}$	$L^\star_{HY}$	$L^\star$
SYS1	25	5.5	6.9	9.912	9.7942	9.2874	9.2656	9.1796	9.1158$^\diamond$
SYS2	35	6.5	7.9	9.623	-	9.1280	8.9301	8.8922	8.7893$^\diamond$
SYS3	40	5.5	6.9	9.473	9.3090	8.9850	8.7178	8.6702	8.5875$^\diamond$
SYS4	45	8.5	9.9	11.086	11.0962	10.8760	10.4042	10.2012	10.0947$^\diamond$
SYS5	50	8.5	9.9	11.646	11.6211	11.3494	10.9865	10.8954	10.7091$^\diamond$
SYS6	60	8.5	9.9	12.842	12.7215	12.3745	11.8399	11.7943	11.7176$^\diamond$
Average				10.7637	10.9084	10.3334	10.0240	9.9388	**9.8357**

Table 5.3: Behavior of GDSBH on instances of Set1. The symbole "$-$" (resp. "$\diamond$") means that the value is not available (resp. corresponds to the best solution realized by the corresponding algorithm) for the corresponding instance.

Columns from 1 to 4 of Table 5.3 show the instance's information (the instance label, its size n and its corresponding height H and width W). Column 5 shows the length $L^\star_{\text{SYS}}$ obtained by SYS. Column 6 reports BSAs' lengths ($L^\star_{\text{BSA}}$) whereas column 7 (resp.

column 8) displays the length $L^{\star}_{\text{KBTG}}$ (resp. $L^{\star}_{\text{KBTG}_p}$) realized by the sequential (resp. parallel) algorithm KBTG (In this case, as indicated by the authors, no-limit has been fixed for the parallel version of the algorithm (see Kubach *et al.* [45]). Column 9 shows the best HY's objective value (among the three values extracted from Hifi and Yousef [38]). Finally, column 10 displays the length $L^{\star}$ obtained by GDSBH.

The percentage improvement, when it happens, yielded by GDSBH according to the results reported in Table 5.3, are summarized in Table 5.4. The last table is composed of two parts: the first part report the instance's information and the second one displays the percentage improvement realized by GDSBH over the other algorithms (noted %SYS, %BSA, %KBTG$_s$, %KBTG$_p$ and $\%HY$, respectively). We recall that these reported values are associated to the runtime limit fixed to one hour.

	#Inst.			GDSBH vs all methods (% Improvement)				
Label	n	H	W	%SYS	%BSA	%KBTG$_s$	%KBTG$_p$	%HY
SYS1	25	5.5	6.9	8.73	7.44	1.88	1.64	0.70
SYS2	35	6.5	7.9	9.49	-	3.85	1.60	1.17
SYS3	40	5.5	6.9	10.31	8.40	4.63	1.52	0.96
SYS4	45	8.5	9.9	9.82	9.92	7.74	3.07	1.06
SYS5	50	8.5	9.9	8.75	8.52	5.98	2,.59	1.74
SYS6	60	8.5	9.9	9.60	8.57	5.61	1.04	0.65
Average				9.44	10.91	5.06	1.91	1.05

Table 5.4: Percentage improvements realized by GDSBH over all other methods on instances of the first set Set1.

From both Tables 5.3 and 5.4, we can remark what follows:

1. First, GDSBH outperforms all tested algorithms (i.e., SYS, BSA, both KBTG$_s$ and KBTG$_p$, and the three versions of HY) since for the instances of Set1 it is able to provide new lower bounds.

2. Second, when comparing GDSBH's results to those reached by SYS, its percentage improvement varies from 8.73% (instance SYS1) to 10.31% (instance SYS3). Globally, the average percentage of improvement on the instances of Set1 is close to 10%.

3. Third, the percentage improvement becomes slightly more interesting when comparing DSBH's results to those realized by BSA. Indeed, in this case, DSBH's percentage improvement varies between 7.44% (instance SYS1) and 9.92% (instance SYS4). Its average percentage improvement is now close to 11%

4. Fourth, The percentage improvement remains positive when comparing GDSBH's results to those provided by both sequential and parallel versions of KBTG algorithm. Indeed, one the one hand, the variation of the percentage improvement of the sequential version of the algorithm remains between 1.88% (instance SYS1) and 7.74% (SYS4). On the other hand, KBTG's parallel version remains less efficient than GDSBH because its global average improvement remains close to 2%.

5. Fifth and last, DSBH's results remain better than those reached by the three versions of the width-beam search HY, since its average percentage improvement remains close to 1%. In this case, the percentage of improvement varies from 0.65% (instance SYS6) to 1.74% (instance SYS5), which remains very interesting for a sequential algorithm using a global strategy combined with both hill-climbing and internal dichotomous search.

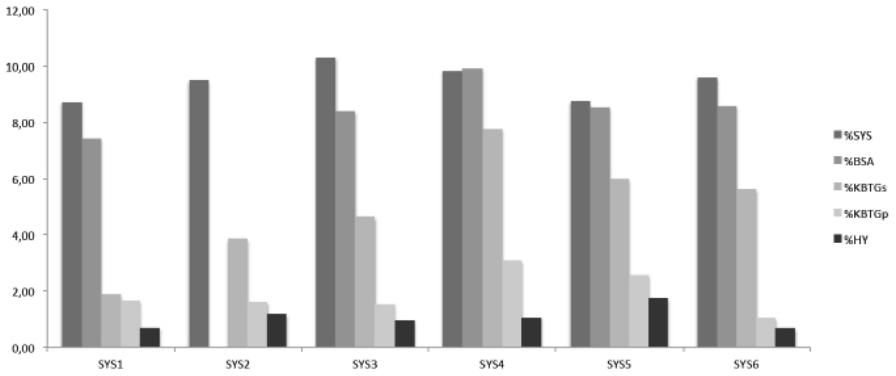


Figure 5.4: Variation of the percentage improvement realized by GDSBH on the instances of the first set Set1.

First, Figure 5.4 shows the behavior of GDSBH on the five instances of Set1: each group represents the variation of the improvements when compared to the results reached by SYS, BSA, $KBTG_s$, $KBTG_p$ and the best results obtained by the three versions of HY, respectively. Second and last, Figure 5.5 shows (on the left hand) GDSBH's best solution reached by GDSBH for the instance SYS6 whereas the right hand of Figure 5.5 reports the best structure realized by GDSBH for the instance SYS5.

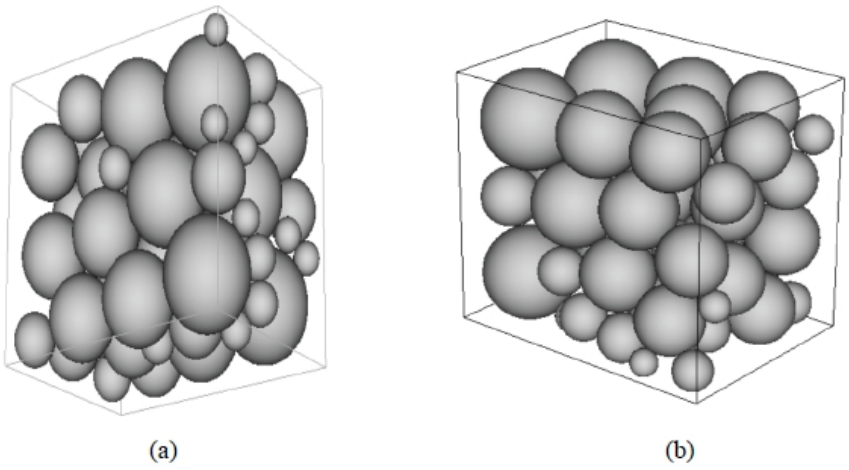


Figure 5.5: Illustration of GDSBH's behavior on SYS6 (on the left hand) realizing the smallest percentage improvement of 0.65%, and on SYS5 (on the right hand) realizing the greatest percentage improvement of 1.74%).

5.3.4 Performance of GDSBH versus KBTG and HY heuristics (Set2)

This section compares the results obtained by GDSBH to those realized by both KBTG$_s$ and HY on instances of Set2 extracted from Kubach *et al.* [45] (since they are the existing algorithms that have tested instances of Set2). For this type of instances, instead of determining the minimum length $L^\star$ of the final container $\mathcal{P}$, Kubach *et al.* [45] published the density of all packed items in the final container. In our case, we also report the lengths (namely $L^\star$) realized by all tested algorithms that correspond to each density.

	#Inst.			KBTG	HY	GDSBH			
Label	n	H	W	$d^\star_{KBTG_s}$	$L^\star_{HY}$	$d^\star_{GDSBH}$	$L^\star_{GDSBH}$	%KBTG	%HY
KBTG1	30	10	10	54.096	10.8076	61.0676	10.7251	11.4162	0.77
KBTG2	30	10	10	30.071	1.99	30.071	1.99	0.0000	0.00
KBTG3	30	10	10	51.387	18.1936	54.3437	18.1093	5.4407	0.47
KBTG7	50	10	10	55.372	12.9653	62.8374	12,9013	11.8805	0.50
KBTG8	50	10	10	45.060	2.5820	60.6938	2.5452	25.7585	1.45
KBTG9	50	10	10	52.732	27.7152	56.9661	27.7152	7.4327	0.00
Average				48.1197	12.3756	**54.3299**	**12.3310**	*11.4307*	**0.53**

Table 5.5: Performance of GDSBH versus both KBTG$_s$ and HY on instances of Set2

All results obtained by the three methods (GDSBH, KBTG$_s$ and HY, respectively) are reported in Table 5.5. Columns from 1 to 4 show the characteristics of each instance. Column 5 reports the best density realized by KBTG$_s$'s algorithm (taken from Kubach *et al.* [45],kubach2011). Columns 6 displays the best final length $d^{\star}_{HY}$ reached by the three versions of HY. Column 7 and 8 reports GDSBH's final density $d^{\star}_{GDSBH}$ and its corresponding length $L^{\star}_{GDSBH}$, respectively. Finally, column 9 (resp. 10) displays the percentage improvement realized by GDSBH when compared to KBTG$_p$ (resp. HY). From Table 5.5, we observe what follows:

1. GDSBH is able to improve most solutions obtained by the sequential version of KBTG. Indeed, on the one hand, it provides five better solutions and it matches the optimal solution corresponding to the instance KBTG2. On the other hand, when the improvement happens, GDSBH realizes an average improvement varying from 5.44% (instance KBTG3) to 25.76% (instance KBTG8). Globally, GDSBH realizes an average percentage improvement of 11.43%.

2. GDSBH has a better behavior when comparing its obtained results to those reached by the three versions of HY algorithm. Indeed, one the one hand, GDSBH confirms its superiority with the parameters $(\rho, \epsilon) = (2000, 20\%)$ by improving all the instances (except for the instance KBTG2 which has been solved by all algorithms to optimality). On the other hand, the improvement gap varies from 0.47% (KBTG3) to 1.45% (KBTG8). In this case, it realizes an average percentage gap of 0.53%, which remains interesting for such a problematic.

5.3.5 Performance of GDSBH on large-scale instances (Set3)

In this section, we evaluate the behavior of the proposed GDSBH on the third set (noted Set3) containing 22 large-scale instances (taken from Hifi and Yousef [38]). Note also that since no codes are available for SYS, KBTG$_s$ and KBTG$_p$, we then compared GDSBHs' solutions to the best solutions reached by the three versions of HY. Because the 22 instances represents large-scale ones, then two runtime limits have been considered in Hifi and Yousef [38]: the first runtime limit t_1 was fixed to one hour (as used in Sections 5.3.3 and 5.3.4) and the second runtime was doubled, i.e., t_2 was fixed to two hours. Herein, we use the same runtimes for comparing the results realized by both methods.

These results are reported in Table 5.6. The first column shows the instance label. The three next columns display, for the first runtime t_1, HYs' solutions, GDSBHs' solutions and the percentage improvement realized by GDSBH versus HY. Finally, for the second runtime limit t_2, the last three columns reports HYs' solutions, those realized by GDSBH and the percentage improvement realized by GDSBH versus HY.

#Inst.	With the runtime limit t_1			With the runtime limit t_2		
	$L^{\star}_{HY}$	$L^{\star}_{GDSBH}$	%Imp.	$L^{\star}_{HY}$	$L^{\star}_{GDSBH}$	%Imp.
SYS.1.3.a	18.0193	17.6541	2.0686	17.6599	17.4765	1.0494
SYS.1.3.b	17.9288	17.8331	0.5366	17.8404	17.8001	0.2264
SYS.1.4.a	34.0449	33.7071	1.0022	33.7266	33.6954	0.0926
SYS.1.4.b	14.5082	14.4541	0.3743	14.2626	14.2002	0.4394
SYS.1.5.a	34.8934	34.8762	0.0493	34.9285	34.8543	0.2129
SYS.1.5.b	15.0323	14.7818	1.6947	14.7812	14.6732	0.7360
SYS.1.6.a	37.7972	37.3114	1.3020	37.3751	37.0124	0.9799
SYS.1.6.b	16.2319	15.9001	2.0868	15.9117	15.7224	1.2040
SYS.2.3.a	15.4148	15.0821	2.2059	15.0985	15.0004	0.6540
SYS.2.3.b	21.2641	21.1228	0.6689	21.1228	21.0011	0.5795
SYS.2.4.a	26.6540	26.4165	0.8991	26.4935	26.3354	0.6003
SYS.2.4.b	15.9796	15.8345	0.9164	15.6993	15.5344	1.0615
SYS.2.5.a	27.5623	27.0778	1.7893	27.0718	27.0002	0.2652
SYS.2.5.b	16.2175	16.1536	0.3956	16.1536	16.1501	0.0217
SYS.2.6.a	29.3127	29.2441	0.2346	29.2633	29.0733	0.6535
SYS.2.6.b	17.4633	17.4224	0.2348	17.1971	17.0225	1.0257
SYS.3.4.a	32.6665	32.5005	0.5108	32.5294	32.1244	1.2607
SYS.3.4.b	14.2483	14.0865	1.1486	14.0963	14.0004	0.6850
SYS.3.5.a	33.9108	33.4542	1.3649	33.7144	33.2046	1.5353
SYS.3.5.b	14.7881	14.2232	3.9717	14.5575	14.2034	2.4931
SYS.3.6.a	36.7125	36.2041	1.4043	36.3645	36.1015	0.7285
SYS.3.6.b	15.8566	15.6591	1.2612	15.6466	15.4358	1.3659
Average	23.0231	**22.7727**	*1.1873*	22.7952	**22.6192**	*0.8123*

Table 5.6: Performance of GDSBH versus HY on the 22 large-scale instances of Set3

From Table 5.6, one can observe what follows:

1. GDSBH realizes better average values. Indeed, for the first runtime limit t_1, fixed to 3600 seconds, it realizes an average value of 22.77 which improves the best average solution values (23.02).

2. Increasing the runtime limit (using $t_2 = 7200$ seconds) for both algorithms (HY and GDSBH), GDSBH realizes 22 new solutions. In this case, the average improvement is more interesting since it becomes equal to 22.62 (instead of the value of 22.80 matched by HY) that represents a percentage of improvement close to 1%.

5.4 Conclusion

In this paper, we investigate the use of the global dichotomous search-based heuristic for approximately solving the three-dimensional sphere packing problem The proposed method is based on the three features: (i) creating a subset of paths by applying an extended greedy procedure, (ii) using a local operator in order to estimate lower bounds associated to internal nodes and (iii) applying an iterative search for diversifying the search process. The performance of the proposed algorithm was evaluated on benchmark instances taken from the literature and the results reached by the method were compared to those reached by recent methods available in the literature. The proposed method succeeded to all methods in yielding new solutions for most of instances.

In the experimental part, in has been noticed that the diversification strategy helped to improve the quality of the solutions obtained by the method. This phenomenon happened especially when the runtime limit has been increased. We believe that a parallel version of the method could be useful for reducing the runtime and increasing the quality of solutions. A parallel cooperative approach seems a good choice, where each processor is assigned a predefined order of items and all processors simultaneously synchronize the solutions reached up to now. In the near future, we will focus to realize such an approach.

Chapter 6

A cooperative method for the sphere packing problem

Contents

In this chapter, we study the three-dimensional sphere packing problem which consists in finding the greatest density of a (sub)set of predefined spheres (objects) into a cube (parallelepiped or closed container). Such a problem is tackled by applying a cooperative method that combines three main features: (i) a best-local position procedure stage, (ii) an intensification stage and (iii) a diversification stage. The first stage ensures a starting feasible solution using a basic greedy local strategy. The second stage tries to solve a series of decision problems in order to place a subset of complementary spheres. The third stage forces the current solution to remove some packed items and then replaced them with other promising spheres. The performance of the proposed method is evaluated on a set of benchmark instances taken from the literature. The results realized by the proposed method are compared to those reached by recent published methods.

6.1 Introduction

Herein, we study a variant of the packing problem which consist in finding the greatest density of any packing of objects into a region of predetermined shape. An instance of the problem is characterized by a cube C (closed container / parallelepiped) of fixed dimensions $L \times H \times W$ and a set N of n spheres (objects) of radii $r_i,\ i \in N$. The goal of the problem is to find the greatest density (occupied volume) of the container C with the available spheres of N. A feasible packing is a configuration of packed spheres without overlapping between spheres and between spheres and the cube.

By using a simple adaptation of the model described in George *et al.* [23] (cf. Hifi and M'Hallah [30]), the problem can be formulated as the following non-linear mixed integer programming:

$$\max \quad \alpha \sum_{i \in N} r_i^3 \xi_i \tag{6.1}$$

$$\text{s.t.} \quad \xi_i r_i \leq x_i \leq \xi_i (L - r_i),\ i \in N, \tag{6.2}$$

$$\xi_i r_i \leq y_i \leq \xi_i (H - r_i),\ i \in N, \tag{6.3}$$

$$\xi_i r_i \leq z_i \leq \xi_i (W - r_i),\ i \in N, \tag{6.4}$$

$$\xi_i \xi_j (r_i + r_j) \leq \sqrt{(x_i - x_j) + (y_i - y_j) + (z_i - z_j)},\ (i,j) \in N^2, i < j, \tag{6.5}$$

$$\xi_i \in \{0,1\},\ i \in N, \tag{6.6}$$

where the objective function (Equation (6.1)) favors the subset of spheres maximizing the density such that $\alpha = (4\pi)/(3LHW)$. Equations from (6.2) to (6.4) ensure that the corresponding sphere i is placed into the container. Equation (6.5) ensures that the non-overlapping between each couple of spheres belonging to the container. Each binary variable $\xi_i,\ i = 1, \ldots, n$, (Equation (6.6) takes the value 1 if the i-th sphere is placed in the container; 0 otherwise. Note that , on the one hand, one can observe that the above model has $n(n-1)/2$ non-convex constraints and $6 \times n$ linear constraints (Equations from (6.2) to (6.4)) according to the binary variables. On the other hand, each of these constraints is related to the binary decision variables, which makes the problem more complex to solve.

We propose to solve the problem by using a cooperative method that combines three main futures: (i) a best-local position procedure, (ii) an intensification strategy and (iii) a diversification strategy. The method begins by building a starting feasible solution by using a greedy procedure that is based on the best local position. The resulting solution will be modified to increase the density of the container occupied by the available spheres. At this stage, solving the current problem is equivalent to solve a decision problem, where the used procedure is able to respond if the container is able to receive the selected spheres without any overlap or not. Such a stage can be viewed as an intensification

strategy, which guides the method to choose more promising solutions. On the other hand, because the studied problem admits an important number of local optima, then given a current feasible solution, the method tries to destroy the solution by removing some positioned spheres from the current container and replacing them by other unpacked ones. The current stage can be viewed as a diversification strategy that is combined with an intensification stage which takes place again in order to improve the provided solution.

The remainder of the chapter is organized as follows. A summarized version of the cooperative method is exposed in Section 6.2, where three stages are used. First, Section 6.2.1 gives the basic procedure that is used for building a starting feasible solution or a complementary solution that completes any partial solution. Second, an adaptation of Wang *et al*'s approach is used as an intensification strategy for packing non-identical spheres into a cube. In this case, new eligible positions are determined according to the current solution, and a series of spheres are positioned at some eligible positions in order to improve the quality of the current solution. Third and last, Section 6.2.3 provides an overview of the cooperative algorithm. Section 6.3 evaluates the performance of the proposed algorithm by comparing its obtained results to those reached by recent algorithms available in the literature. Finally, Section 7.4 concludes the chapter.

6.2 A cooperative method

This section exposes the cooperative method for the sphere packing problem. The main principle of the cooperative method can be summarized as follows:

(i) Starting the search process by an initial solution using a basic greedy procedure (cf. Section 6.2.1).

(ii) Building an improved solution using a series of decision problems (cf. Section 6.2.2).

(iii) Perturbing the search process and re-constructing a new current solution using a basic greedy procedure according to the new order (cf. Section 6.2.3).

(iv) Steps (ii)-(iii) are repeated until a satisfactory solution is reached.

In order to simulate Step (i), the cooperative method starts with a quick greedy feasible procedure. It works by fixing, at each step, a sphere until a final feasible solution of the problem is obtained; that is a starting solution. Improving the quality of a given solution is often related to the search process used around that solution. We do it by applying an adaptation of Wang *et al.*'s [72] approach for solving a series of decision problems. The goal of such an adaptation is to augmente the density of the packed spheres from an existing feasible solution. Finally, using a diversification strategy is often promising to escape from local optima. Hence, the last strategy of the cooperative method tries to

randomly un-assign some spheres from the current solution, and tries to complete it using both the basic greedy procedure and the resolution of a series of decision problems.

In what follows, first, Section 6.2.1 exposes the used greedy procedure in order to yield a feasible solution. Second, Section 6.2.2 gives the intensification stage which tries to improve the solution yielded after each iteration. Third and last, Section 6.2.3 details the diversification stage which tries to explore other local optima and to enhance the solutions built.

6.2.1 Building a feasible solution

Generating a solution is equivalent to generate a subset of positions of the spheres in the current container $\mathcal{P}$. It can be done by applying the so-called *Basic Greedy Procedure* (namely BGP) (cf. Hifi and Yousef [36, 37]) that is based on positioning a series of spheres step by step following the best local position of all candidate spheres. Such a procedure can also be used for building either a partial solution or a complementary one.

Let suppose that all spheres are reorganized in decreasing order of their radii. Assume that the bottom-left-depth corner of $\mathcal{P}$ is positioned at $(0, 0, 0)$, where $\mathcal{P}$ is represented by a set of faces $\mathbb{F} = \{\text{left}, \text{top}, \text{right}, \text{bottom}, \text{depth}, \text{front}\}$. Then, $\mathcal{P}$ is represented in the Euclidian space, such that:

- Each sphere $i \in N$ is centered at the position (x_i, y_i, z_i).
- A pair (i, j) of N are represented by their distance $\delta_{i,j}$:

$$\delta_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} - (r_i + r_j), \quad \forall (i, j) \in N^2, \tag{6.7}$$

where both i and j can be positioned when $\delta_{i,j} \geq 0$ and constraints from (6.2) to (6.4) are satisfied.

Herein, a partial solution is equivalent to a subset of positions in the current container $\mathcal{P}$. Let $i \in N$ be the current sphere to be packed into $\mathcal{P}$ and P_{I_i} be the subset of its eligible positions. The first sphere of N is packed at the position (r_1, r_1, r_1) and for $i \in N,\ i \geq 2$, the following notations are considered:

- I_i: the set of spheres of N already packed in the current container $\mathcal{P}$.
- $\overline{I_i}$: the set of spheres of N which are not yet packed in the current container $\mathcal{P}$.
- P_{I_i}: the set of distinct eligible positions for the next sphere i to pack given the set of packed spheres I_i.

It follows that an eligible position $p_{i+1} \in P_{I_i}$ (for the sphere i) is that touching three different elements $e_1,\ e_2$ and e_3. An element is either a sphere of N already positioned (representing I_i) or one of the six faces belonging to $\mathbb{F}$. So, assume that $\mathcal{T}_{p_{i+1}}$ denotes the

set of these positions according to e_1, e_2 and e_3. Then, a step of BGP, for $i \geq 2$, $i \in N$, proceeds as follows: the distance $\Delta(p^k_{i+1})$ characterizing the $(i+1)$-th sphere to pack, when positioned at the eligible position $p^k_{i+1} \in P_{I_i}$, realizes what follows:

$$\Delta(p^k_{i+1}) = \min_{j \in I_i \cup \mathbb{F} \setminus \mathcal{T}_{p^k_{i+1}}} \delta_{(i+1,j)}. \tag{6.8}$$

Hence, BGP starts by positioning the first sphere $i = 1$ at the bottom-left-depth position, i.e., at the position (r_1, r_1, r_1), while the remaining $n-1$ spheres are successively positioned according to $\Delta(.)$ (cf., Equation (6.8)). Note that if different eligible positions have the same distance, then one can choose either (i) the first sphere according to the order initially fixed or (ii) the sphere that realizes the smallest distance to the left side of $\mathcal{P}$. In what follows, the framework of the the basic greedy procedure, used for reaching a starting feasible solution, is described.

1. Let $I_i = \{i\}$, $\overline{I_i} = \{1, .., n\} \setminus \{I_i\}$ and P_{I_i} be the set of eligible positions of the i-th sphere.
2. Update P_{I_i} with the eligible positions of each $j \in \overline{I_i}$.
3. Repeat
   (a) Let p^{i+1}_k be the best position for the next sphere according to Equation (6.8).
   (b) Position the $(i+1)^{th}$ sphere into p^{i+1}_k and move the packed sphere to I_i.
   (c) Update the set of eligible positions P_{I_i}.
   Until $P_{I_i} = \emptyset$
4. Exit with a feasible solution of density D and the subset $I_i \subseteq N$ containing the packed spheres.

BGP can be viewed as a greedy procedure that tries to pack all spheres into the container $\mathcal{P}$. Whenever all the spheres are placed successfully, then an optimal solution is reached for the problem. Otherwise, a subset of spheres is placed with a density of D. Indeed, according to the instances of the literature, when can observe that in some cases the instances used may be very easy to solve instead of other ones for which computing good approximations remain very difficult.

Indeed, in order to show the hardness of some used instances and how BGP can fill the container, two instances taken from the literature (KBG2 with 30 spheres to pack and SYS1KP with 25 spheres to pack) are considered for illustrating the phenomenon. Figure 6.1 illustrates KBG2's solution provided by BGP whereas Figure 6.2 shows that of SYS1KP. As one can see, the greedy procedure is able to pack all spheres (cf. Figure 6.1) realizing a density of 30.0709% (that is an optimal solution) whereas it packs 12 spheres over 25 for the second instance (cf. Figure 6.2) by realizing a density of 48.4266% (that is an approximate solution).

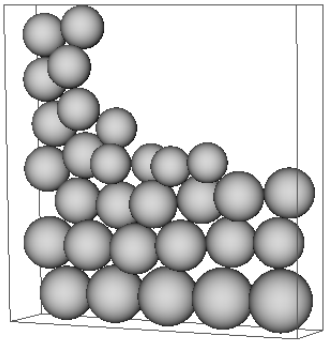


Figure 6.1: Illustration of BGP's solution for the instance KBG2 containing 30 spheres (the density of the provided solution is equal to 30.0709%).

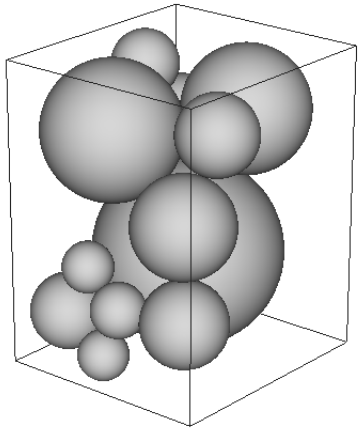


Figure 6.2: Illustration of BGP's solution for the instance SYS1KP containing 25 spheres (the final solution realizes a density of 48.4266%).

6.2.2 Improving the quality of solutions

Solving a combinatorial problem often involves methods based upon enumerative searches. For the studied problem, a series of subset of spheres may provide several feasible configurations with the same density. In some cases, the same (sub)set of spheres may also realizes the same density whereas the positions of the spheres on the container $\mathcal{P}$ are definitely different. These solutions may correspond to local optima where most of the algorithms used stop with such kind of solutions.

In order to escape from these solutions, Wang *et al.* [72] proposed a quasi-physical quasi-human algorithm, which simulates the physical model for tackling the circular packing (packing a set of circles into a containing circle of minimum radius). The approach is composed of two stages. The first stage applies the quasi-physical approach that solves a decision problem which checks if a feasible packing of the available circles in the containing circle is possible. The quasi-human approach is then employed to make jumps in order to get out of local minima. Because the used process is often trapped in the infeasible local optima, circles are allowed to locate new positions according to the packed circles. The second stage recalls the first stage after reducing the radius of the containing circle. Such process is iterated till obtaining the final containing circle of minimum radius.

Herein, we use an adaptation of Wang *et al*'s approach as a part of the cooperative method for packing non-identical spheres into a cube. In our study, the proposed adaptation can be viewed as two-step procedure:

(i) searching for new eligible positions according to the current solution, and

(ii) trying to add a series of spheres to some eligible positions by solving a series of decision problems.

Let S be the current solution (not necessary the starting solution reached by BGP). Suppose that $N_1 \subseteq N$ be the subset of spheres positioned into the container $\mathcal{P}$ and $N_2 = N \setminus N_1$ be the remaining non-packed spheres. Then, the following description explains the two-steps procedure used for determining a better feasible solution when it exists.

A. The first step.

From the current solution, containing the set N_1 of positioned spheres, one can determine a set of eligibles positions for the non-packed spheres. It can be obtained by applying the strategy used by the basic procedure: to determine some positions inside the container, where for a candidate sphere i, its eligible position p_{i+1} is that touching three elements e_1, e_2 and e_3. Each element is either a sphere of N already positioned (representing the set I_i) or one of the six faces belonging to the set $\mathbb{F}$. We then apply the following steps in order to compute the current eligible positions for all remaining spheres:

1. Let r_{max} (resp. r_{min}) be the greatest (resp. smallest) radius among the unpacked spheres.
2. Let r_f be the radius of the "*fictive sphere*", which is initialized to r_{max} and $L_p = \emptyset$ be the set of preselected positions.
3. While $\left(r_f > r_{min}\right)$ do
   (a) Compute all eligible positions according to the "*fictive sphere*";
   (b) Update L_p and reduce the radius of the current "*fictive sphere*".

B. The second step.

Determining an improved solution (with better density) is equivalent to solve a decision problem. Indeed, given a subset N_1 of spheres of radius r_i, $i \in N$, and a container of dimensions (L, W, H), is it possible to pack all the spheres inside the cube or not. Such a problem, noted DP_{N_1}, can be formulated as follows:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{|N_1|} \sum_{j=i+1}^{|N_1|} O_{i,j}^2 + \sum_{i=1}^{|N_1|} (O_{i,x} + O_{i,y} + O_{i,z}) \\
\text{s.c.} \quad & O_{i,j} = \max\left\{0, r_i + rj - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}\right\}, \ i \neq j, i, j \in N_1^2 \\
& O_{i,x} = \max\left\{0, r_i + |x_i| - 0.5L\right\}, \ i \in N_1 \\
& O_{i,y} = \max\left\{0, r_i + |y_i| - 0.5H\right\}, \ i \in N_1 \\
& O_{i,z} = \max\left\{0, r_i + |z_i| - 0.5W\right\}, \ i \in N_1.
\end{aligned}
$$

The problem DP_{N_1} is a continuous local optimization, which admits a local optimum whenever its objective function is equal to zero. It means that all spheres of N_1 can be packed into the container. Otherwise, the procedure fails to provide a feasible solution for the current subset of spheres N_1.

Algorithm 16 describes the main steps of the Insertion Procedure (noted IP) which is applied for yielding an extended (better) feasible solution for the problem. It can be viewed as a stepwise insertion procedure where, at each step of the procedure, the insertion respects the feasibility of the provided solution; that is equivalent to solve the decision problem DP_{N_1} (described above).

Algorithm 16 . Insertion Procedure (IP)

Input. A starting feasible solution S with a subset N_1 (resp. N_2) of packed (resp. non-packed) spheres.

Output. An (improved) solution $S^\star$ and (updated) subsets N_1 (resp. N_2) of packed (resp. non-packed) spheres.

1: Set $\overline{N}_2 = N_2$ and i be the first item of $\overline{N}_2$ (according to the current order).
2: **while** ($\overline{N}_2 \neq \emptyset$) **do**
3: Set $O = N_1 \cup \{i\}$ and solve the decision problem DP_O.
4: **if** (all spheres of O are packed) **then**
5: Let S' be the new solution.
6: update $S^\star$ with S' (of density $D^\star$).
7: Set $N_1 = O$.
8: **end if**
9: Set $\overline{N}_2 = \overline{N}_2 \setminus \{i\}$.
10: **end while**
11: **Exit** with the best solution $S^\star$ and the updated subsets N_1 and $N_2 = N \setminus N_1$.

We recall that IP applies a sequentially insertion according to the available spheres (belonging to the subset N_2), one by one, into the existing solution. The main loop (Lines from 2 to 10) stops when all the spheres have been examined. Line 3 serves to solve a decision problem DP_O (by applying Wang *et al.*'s procedure), where a new subset of spheres is considered (that is composed of the subset N_1 augmented with a new sphere belonging to the complementary subset N_2 − initialized to $\overline{N}_2$ for giving a chance to each sphere initially belonging to N_2). Lines 6 and 7 check the feasibility of the new solution and update that solution with the new positioned spheres in the case where the decision problem DP_O is true. Line 9 decreases the size of the set of un-packed spheres. Such a process is repeated till examining all spheres initially unpacked. Finally, the procedure exits (line 11) with the best solution $S^\star$ (with density $D^\star$) and both updated subsets of packed and unpacked spheres.

In order to illustrate IP's mechanism, Figures 6.3 and 6.4 display two solutions. The first solution (cf. Figure 6.3) is constructed by using the greedy procedure BGP and the second one (cf. Figure 6.4) is obtained after running IP. One can observe that despite the good result that can reach BGP, the used process improves the quality of the solution. Indeed, the density of the new solution reached by IP becomes equal to 51.0489% instead of 48.4266%.

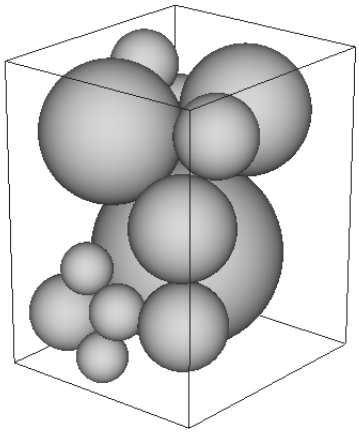


Figure 6.3: BGP's solution for the instance SYS1KP with a density equals to 48.4266%, representing 12 positioned spheres.

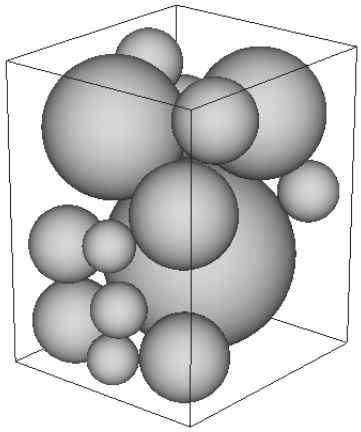


Figure 6.4: Illustration of the solution provided by the second step of insertion procedure: the density of the solution is augmented to 51.0489%,representing 14 positioned spheres.

6.2.3 Using a diversification strategy

IP tries to find a series of feasible solutions of the problem, which are often considered as local optima. The objective of the building procedure is to provide a series of neighborhoods, issuing from the solution at hand, which might contain better solutions. Despite some improvements that can be realized, and because of the number of achievable solutions with the same objective value, it is interesting to see how diversifying the search process through other solutions.

Herein, we propose a diversification strategy that consists of removing a subset of spheres from the current solution (i.e. a feasible solution of the problem). The removing strategy diversifies the search process by degrading the quality of the solution at hand with the aim of avoiding stagnating in a local optimum. Then, a partial solution is obtained and it is completed by applying the basic procedure BGP as a tool for refining the quality of the current partial solution, according to the new order associated to the remaining spheres. Such a strategy has been already used with success for solving variants of the knapsack type problems (cf., Hifi [28] and, Hifi and Michrafy [33]).

The main principle of the diversification procedure is described in Algorithm 17. It starts (Line 2) by dropping the quality of the current solution: destroying the current solution (noted S^*), where $\alpha\%$ of spheres belonging to S^* are randomly removed. Then, a new order is reconsidered (Line 3) by positioning $\alpha\%$ of spheres after the first unpacked sphere; that is the first sphere not positioned in the current solution according to the current order. A new (feasible) partial solution S^p is reached (Line 4) which is completed by

Algorithm 17 . Drop and Rebuild Procedure (DRP)

Input. S, a feasible solution with its subset N_1 (resp. N_2) of packed (resp. non-packed) spheres.

Output. S', an (improved) solution and its (updated) subsets N_1 (resp. N_2) of packed (resp. non-packed) spheres.

1: Set $S^\star \longleftarrow S$;
2: Apply a random destroying strategy (remove $\alpha\%$ of packed spheres) from the solution $S^\star$.
3: Place the $\alpha\%$ of spheres after the first unpacked sphere.
4: Let S^p be the new provided partial solution.
5: Complete S^p by using BGP and let S' be the new solution reached.
6: **Return** the solution S' (with density D') and the updated both subsets N_1 and $N_2 = N \setminus N_1$.

applying the basic procedure BGP (cf. Section 6.2.1). Because a new order is considered, then the procedure tries to build (Line 5) a new feasible solution, noted S'. Finally, the procedure returns the resulting solution S' and its updating subsets N_1 and N_2.

Note that, the strategy used for forwarding the removed spheres may be considered as a flexible diversification. It tries to give a chance to the first un-positioned sphere since it was not previously inserted. Another strategy may be considered by transferring all removed spheres after the unpacked spheres. In this case, such a strategy can be seen as an aggressive diversification which would completely change the search space. Limited computational results showed that the strategy currently used (the first one) was able to produce solutions with good quality within lesser runtime.

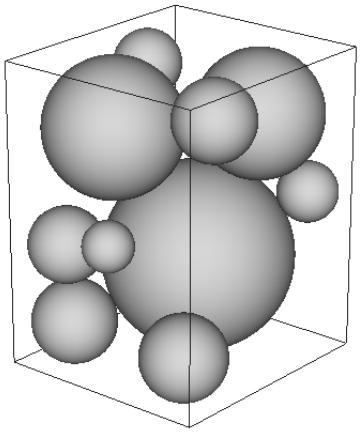


Figure 6.5: Using the first stage of the diversification strategy: the dropping stage (the solution realizes a density equals to 45.021%, where 30% of packed spheres were removed from the current solution).

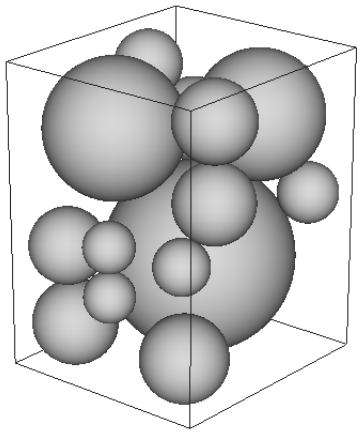


Figure 6.6: Using the second stage of the diversification strategy: the rebuilding stage (a solution realizing a density of 49.3742%).

Figures 6.5 and 6.6 illustrate (i) the partial solution realized when the dropping stage (removing 30% of spheres from the packed ones) is applied, and (ii) the final solution obtained when the rebuilding stage is applied. As shown from Figure 6.5, the first solution degrades the quality of the solution as expected whereas the second solution (cf. Figure 6.6) seems as a modified solution. If such a solution is compared to that builded by BGP (cf. Figure 6.3), one can observe that both solutions differ.

6.2.4 An overview of the cooperative method

Algorithm 18 summarizes the main steps of the proposed Cooperative Method (noted CM). The input of CM is an instance of the packing problem composed of a set N of n spheres that would be positioned into a cube of dimensions (L, W, H). It starts (Line 1) by ordering the spheres of the set N in decreasing order of their radii. At line 2, the first initial feasible solution S' is reached by applying the basic procedure BGP (cf. Section 6.2.1). Lines from 4 to 15 represent the main loop of CM, which is composed of both intensification and diversification stages.

The intensification stage (Lines from 5 to 8) applies IP procedure in order to solve a series of decision problems. We recall that the decision problem amounts to find the best density of spheres regarding the (current) specific order of the spheres. Whenever the new solution achieves a better density so far, then the best solution $S^\star$ is updated with its subsets N_1 (the packed spheres) and N_2 (the remaining spheres).

The diversification stage (Lines from 10 to 14) applies DRP procedure which combines the destroying operator and the rebuilding strategy. Indeed, the first step removes $\alpha\%$ of packed spheres from the current solution S' and the second one rebuilds a new diversified solution; that is a solution completing the partial solution obtained after destroying the current solution. The complete solution is obtained by using the basic procedure BGP.

In the case where the solution is improved (Lines from 12 to 14), i.e., reaching a higher density, the best solution $S^\star$ is replaced by S_{new} with its two subsets. Because the method alternates between these two stages, in this case the diversified solutions undergo the improvement stage (applying IP procedure), i.e., solving a series of decision problems to attempt improving the quality of the solutions. Finally, both stages are repeated until a satisfactory solution is obtained or after a maximum number of iterations (or fixed runtime) is performed.

In order to show how CM works, Figure 6.7 illustrates (from the left-hand to the right-hand) (i) the solution realized when the rebuilding stage is applied, (ii) the fictive positions associated to the remaining (non-positioned) spheres and, (iii) the final solution obtained when the insertion procedure IP is applied. As shown from Figure 6.7, the density of the new solution becomes equal to 52.1885%.

Algorithm 18 : A Cooperative Method for the sphere packing problem

Input: An instance of packing problem.
Output: $S^{\star}$, the best solution of the problem.

1: Consider that all spheres of N are ordered in decreasing order of their radii.
2: Let S' be a starting solution realized by the procedure BGP (cf. Section **??**).
3: Affect S' to $S^{\star}$ `// the best solution found so far`.
4: **while** (the stopping criteria is not performed) **do**
5: Apply IP to S' and let S_{new} be the new reached solution. `// Improvement stage`
6: **if** ($D_{S_{new}} > D_{S^{\star}}$) **then**
7: Affect S_{new} to $S^{\star}$ and update both subsets N_1 and N_2 (according to S_{new}).
8: **end if**
9: Affect S_{new} to S'.
10: Apply DRP to S' (cf. Section 6.2.2). `// Diversification stage`
11: Let S_{new} be a new solution reached.
12: **if** ($D_{S_{new}} > D_{S^{\star}}$) **then**
13: Affect S_{new} to $S^{\star}$ and update both subsets N_1 and N_2 (according to S').
14: **end if**
15: **end while**
16: **return** $S^{\star}$.

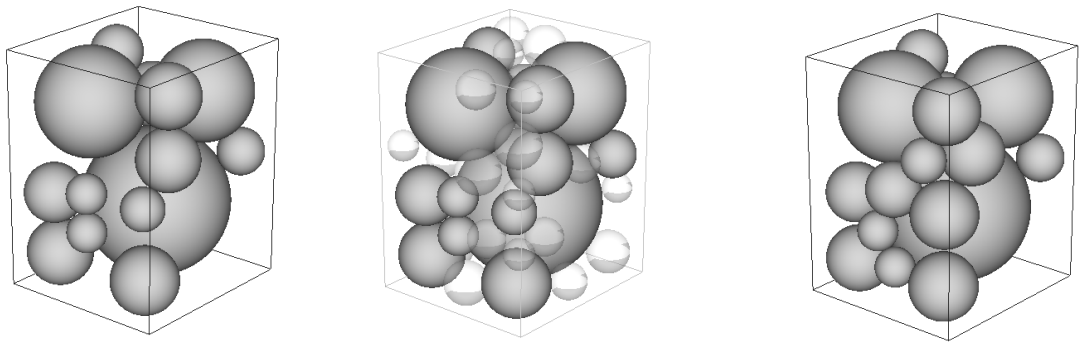


Figure 6.7: Illustration of CMs' iterations on SYS1KP instance: (i) the rebuilding's solution (on the left-hand), (ii) the eligible positions generated according to the unpacked spheres (on the middle) and (iii) the final solution achieved when applying the insertion procedure (on the right-hand).

6.3 Computational results

The purpose of this section is two-fold: (i) to show how to determine a good trade-off between the quality of the obtained solutions when varying the destroying parameter and (ii) to evaluate the effectiveness of the proposed Cooperative Method (CM) when its achieved results are compared to the best results available in the literature. This part was conducted on two sets of instances: the first set (noted Set1) contains twelve instances divided into two subsets, where each of them is composed of six instances. The first subset (noted Set1a) contains instances taken from Akeb [2], where the number of spheres to pack varies from 25 to 60 (noted from SYS1KP to SYS6KP). The second subset (noted Set1b) contains instances (noted KBG1, KBG2, KBG3,KBG7, KBG8 and KBG9) taken from Kubach *et al.* [46], where the number of spheres varies from 30 to 50. These instances proposed and used by Kubach *et al.* [46] and they were used as benchmarks in Akeb [2] for testing his proposed approach. The second set of instances (Noted Set2) is composed of 96 instances extracted from Kubach *et al.* [45]. In this case, Set2 wa also divided into four subsets (noted set2a, set2b, set2c and set2d) following the number of spheres to pack which varies in the discrete interval $\{20, 30, 40, 50\}$.

Note also that because the size of instances differs for both sets, i.e., Set1 and Set2, we then fixed the runtime limit, considered as the stopping criterion of the method, to 3600 seconds for instances of Set1 and 7200 seconds for the instances of Set2 (which represent more hardness instances). Finally, the proposed method was coded in C++ and tested on a Pentium 2.4 Mhz (with 2 Gb of RAM) and the other tested methods were also ran on equivalent computers.

6.3.1 Effect of the parameter α

It is well-known that different parameter settings for any method lead to results of variable quality. In the preliminary results, two parameters should be taken into account by CM (cf., Algorithm 18): the value associated to α and the maximum runtime fixed (which can be considered as a standard runtime limit considered by algorithms of the literature). In the preliminary results, we evaluated the effect of the parameter α, especially for determining the right value. Herein, CM was run on instances of Set1a by varying the value of α in the discrete intervalle $\{10, 20, 30, 40, 50, 60\}$, where for each tested instance, ten trials were considered.

Over the ten trials, Table 6.1 exposes for each tested instance the minimum (Min), the average (Mean) and the maximum (Max) density (noted D) reached by CM regarding the value associated to α. It also shows the number of spheres packed (noted #d) by CM (minium, average and maximum values, respectively). From Table 6.1, one can observe that CM is able to provide better average solution values for $\alpha = 30$. We believe that the phenomenon is due to the greedy procedure that uses an intuitive packing which, in some cases, it induces solutions with lesser densities. Indeed, both smallest and greatest values

	$\alpha = 10$						$\alpha = 20$					
	Min		Mean		Max		Min		Mean		Max	
Inst.	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d
SYS1	52.623	15	53.2558	16.5	53.901	17	52.761	17	53.4298	17.1	53.901	17
SYS2	53.622	25	55.2099	24.7	56.026	26	54.353	24	55.7760	24.8	56.322	24
SYS3	53.511	24	55.1969	25.7	56.252	25	55.113	27	56.1647	26.2	56.449	27
SYS4	52.678	27	53.2778	28.5	53.631	31	53.471	31	53.5515	29.5	53.611	29
SYS5	53.314	29	54.1031	29.4	54.852	30	54.664	29	55.5279	32	56.808	33
SYS6	55.183	35	56.2595	36.2	56.906	35	56.196	37	57.0971	37	57.964	38
G-Av.	53.489	25.83	54.551	26.83	55.261	27.33	54.426	27.50	55.258	27.77	55.843	28.00
	$\alpha = 30$						$\alpha = 40$					
	Min		Mean		Max		Min		Mean		Max	
Inst.	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d
SYS1	52.547	17	53.5928	17.2	54.229	18	53.008	17	53.5395	17	53.653	17
SYS2	54.172	24	55.8714	24.3	56.396	25	54.099	21	55.7458	24.5	56.301	25
SYS3	55.265	27	56.0701	26.8	57.191	26	54.308	23	55.8985	25.7	56.871	26
SYS4	54.538	30	54.7868	30.3	55.734	31	53.035	30	54.2323	30.1	55.856	31
SYS5	54.322	33	56.1864	33.3	57.307	34	55.081	34	56.0719	33	57.012	34
SYS6	56.963	36	57.7119	36.4	58.205	35	55.573	35	57.3126	36.8	58.131	38
G-Av.	54.635	27.83	55.703	28.05	**56.510**	28.17	54.184	26.67	55.467	27.85	56.304	28.50
	$\alpha = 50$						$\alpha = 60$					
	Min		Mean		Max		Min		Mean		Max	
Inst.	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d	D_{CM}	#d
SYS1	50.476	15	51.7235	15.2	52.665	16	50.031	14	50.8633	14.4	51.932	14
SYS2	52.345	21	53.8049	22.5	54.508	25	52.557	22	53.7337	22.8	54.316	22
SYS3	54.025	24	54.5342	24.3	55.368	27	53.143	23	54.1650	24	55.074	24
SYS4	52.294	27	53.1705	28.8	54.445	31	52.222	27	52.7863	26.8	53.448	30
SYS5	53.069	28	54.0218	29.8	55.711	32	53.437	29	54.0989	29.8	55.601	32
SYS6	54.945	35	56.122	35.5	57.839	36	53.858	34	55.5016	34.9	56.249	36
G-Av.	52.859	25.00	53.896	26.02	55.089	27.83	52.541	24.83	53.525	25.45	54.437	26.33

Table 6.1: Behavior of the cooperative method (CM) when varying the parameter α

of α may provide a premature convergence of the method and so, the insertion procedure may have a smallest chance to improve the quality of the solution at hand. Finally, the moderate value of α seems more interesting when combining both the greedy strategy and the insertion one. Hence, α with the value 30 will be adopted for the rest of the paper.

6.3.2 Behavior of CM on instances of Set1

CM is first tested on the set Set1 containing 12 instances. It contains six instances (noted Set1a) taken from [2], where the number of spheres to pack varies from 25 to 60 (these instances are noted from SYS1KP to SYS6KP). The second part (noted Set1b) contains six instances extracted from Kubach *et al.* [46], where the number of spheres varies from 30 to 50 (these instances are noted from KBG1 to KBG9 and used as benchmarks in Akeb [2]).

6.3.2.1 Behavior of CM on instances of Set1a

Table 6.2 compares the performance of CM to that of TSLA (proposed in Akeb [2]). Columns from 1 to 5 indicate the instance label and its related information. Column 6 (resp. column 7) shows D_{TSLA} (resp. $\#d_{TSLA}$), the density (resp. number) obtained (resp. packed) by TSLA whereas columns from 8 to 11 show the same results achieved

									Cooperative Method			
					TSLA		Mean		Max		%Improvement	
Inst.	n	H	W	L	D_{TSLA}	$\#\mathrm{d}_{TSLA}$	D_{CM}	$\#\mathrm{d}_{CM}$	D_{CM}	$\#\mathrm{d}_{CM}$	Mean	Max
SYS1KP	25	6.9	5.5	5	53.008	15	*53.593*	17.2	**54.229**	18	1.091	2.252
SYS2KP	35	7.9	6.5	5	55.358	25	*55.871*	24.3	**56.396**	25	0.919	1.841
SYS3KP	40	6.9	5.5	5	55.086	23	*56.070*	26.8	**57.191**	26	1.755	3.681
SYS4KP	45	9.9	8.5	5	54.048	29	*54.787*	30.3	**55.734**	31	1.349	3.025
SYS5KP	50	9.9	8.5	5	55.757	32	*56.186*	33.3	**57.307**	34	0.764	2.705
SYS6KP	60	9.9	8.5	5	56.960	36	*57.712*	36.4	**58.205**	35	1.303	2.139
G-Av.					55.036	26.667	*55.703*	28.050	**56.510**	28.167	1.197	2.607

Table 6.2: Performance of CM versus TSLA on instances of Set1a. The symbol "$\star$" means that method achieve a better bound, the value in "italic" means that method improves the best solutions of the literature and the value in the "boldface" denotes a new (average) bound.

by CM, i.e., D_{CM} and $\#\mathrm{d}_{CM}$, respectively. Finally, column 12 (resp. column 13) tallies the maximum (resp. average) percentage improvement (when it happens) yielded by CM when compared its results to those reached by TSLA (the percentage improvement is computed as follows: $\frac{D_{CM}-D_{TSLA}}{D_{CM}} \times 100$).

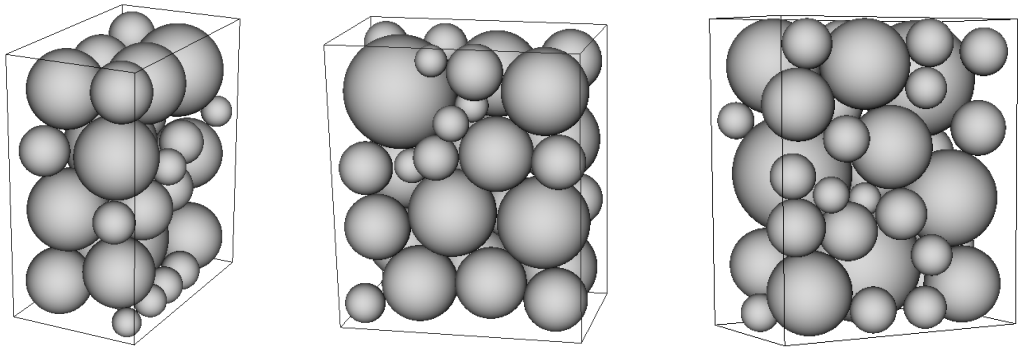
The analysis of Table 6.2 shows what follows:

- CM outperforms TSLA whenever the runtime limit was fixed to 3600 seconds. Indeed, it is able to improve all the bounds for all instances of Set1a.

- When comparing the maximum global average density (cf. the global average value noted G-Av., column 10), one can observe that CM realizes a value of 56.510% which dominates that achieved by TSLA (55.036%). On can also observe that packing more spheres into the container does not automatically lead to a better solution. Indeed, such a phenomenon can be observed on the the instance SYS6KP, where TSLA packed 36 spheres for realizing a density of 56.960% whereas CM positioned 35 spheres for giving a density of 58.205%.

- When comparing the average values achieved by CM (55.703%), one can also observe that CM's G-Av. (column 8) remains superior than that realized by TSLA (55.036%). As discussed above, for the same instance SYS6KP, CM's average density (57.712%, column 8) remains significantly greater than that of TSLA (56.960%, column 6) and it is able to pack 36.4 spheres, on average.

- Furthermore, as shown from column 13, the maximum percentage improvement (Max) is very impressive for such a problem, since it varies from 1.841% (instance SYS2KP) to 3.681% (instance SYS3KP). Note also that even if one consider the average values related to the ten trials (cf., column 12, Mean), we can see the superiority of CM since it is able to realize an average percentage of improvement varying from 0.764 (instance SYS5KP) to 1.755 (instance SYS3KP) with a global average value equals to 1.197.



(a) SYS4KP (b) SYS5KP (c) SYS6KP

Figure 6.8: Illustration of the solutions' structures of three instances (SYS4KP on the left-hand, SYS5KP on the middle and SYS6KP on the right-hand) of the set Set1a.

Finally, Figure 6.8 illustrates three final configurations related to the instances SYS4KP (on the left-hand of the figure), SYS5KP (on the middle of the figure) and SYS6KP (on the right-hand of the figure). These structures represent new solutions, where D_{CM}(SYS4KP) = 55.856%, D_{CM}(SYS5KP) = 57.307% and D_{CM}(SYS6KP) = 58.205%.

6.3.2.2 Behavior of CM on instances of Set1b

Herein, the performance of CM is evaluated on instances of the second part of Set1 (noted Set1b) that contains 6 instances (taken from Kubach *et al.* [46]). Its performance is compared to those realized by the three heuristics: (i) sequential version of B1.16 heuristic with the parameter $\tau = 0.8$ (cf. Kubach *et al.* [46]), (ii) sequential version of B1.16 heuristic with the parameter $\tau = 1.0$ (cf. Kubach *et al.* [46]) and, (iii) TSLA (cf. Akeb [2]). We also evaluated the ratios between CM and the three considered heuristics, i.e., both sequential versions of B1.16 and TSLA, respectively.

					B1.16		TSLA		CM Mean		CM Max	
Inst.	n	L	H	W	D_{v1}	D_{v2}	D_{TSLA}	$\#d_{TSLA}$	D_{CM}	$\#d_{CM}$	D_{CM}	$\#d_{CM}$
KBG1	30	11.103	10	10	53.657	55.001	55.001	30	54.663	29.6	55.001	30
KBG2	30	1.99	10	10	30.071	30.071	30.071	30	30.071	30	30.071	30
KBG3	30	17.785	10	10	52.154	52.375	52.625	28	52.220	27.5	**53.507**	29
KBG7	50	13.71	10	10	55	55	55	50	55	50	55	50
KBG8	50	2.207	10	10	46.406	46.517	46.342	41	46.184	35.8	**47.416**	41
KBG9	50	27.965	10	10	52.54	53.173	53.662	48	52.968	46.5	53.662	48
G-Av.					48.305	48.690	48.784	37.83	48.518	36.57	**49.110**	38

Table 6.3: Performance of CM versus two versions of B1.16 (with $\tau = 0.8$ and $\tau = 1.0$) and TSLA on instances of Set1b

First, Table 6.3 shows the results achieved by both versions of B1.16, TSLA and those

reached by the proposed CM. Columns from 1 to 5 show the instance's information. The results of both versions of B1.16 are displayed in column 6 for the first version (noted D_{v1}) and in column 7 for the second version (noted D_{v2}), respectively. Column 8 tallies those achieved by TSLA (represented by the density D_{TSLA}) and column 9 shows the number of spheres packed by the same heuristic (noted $\#d_{TSLA}$). Finally, columns 10 and 11 (resp. columns 12 and 13) display the average densities (resp. the average number of packed spheres) achieved by CM for the ten trials.

Second and last, in order to show the behavior of CM versus both versions of B1.16 and TSLA, we also evaluated the percentage ratios between CM and B1.16 and, CM and TSLA; both ratios represent $\frac{D_{CM}-B1.16}{D_{CM}} \times 100$ and $\frac{D_{CM}-D_{TSLA}}{D_{CM}} \times 100$, respectively. Table 6.4 reports the percentage ratios relating the quality of the solutions achieved by CM (column 1 for CM versus the first version of B1.16, column 2 for CM versus the second version of B1.16 and, column 3 for CM versus TSLA).
The analysis of the results of both Tables 6.3 and 6.4 follows:

- First, CM outperforms the sequential B1.16 heuristic (with $\tau = 0.8$). Indeed, CM realizes a percentage average value of 49.110% (column 12 and the last line of the table noted G-Av.) compared to the average value of 48.3047% (column 12 and the last line of the table noted G-Av.) realized by the sequential heuristic B1.16. Indeed, CM dominates B1.16 heuristic (with $\tau = 0.8$) in four cases and matches the solutions for the rest of the instances. In this case, the percentage improvement (%Imp.) varies from 2.091% (KPG9) to 2.529% (KPG3) and both CM and B1.16 match the optimal densities for two instances (KBG2 and KBG7).

	CM versus B1.16 ($\tau = 0.8$)		CM versus B1.16 ($\tau = 1.0$)		CM versus TSLA	
Inst.	Mean	Max	Mean	Max	Mean	Max
KBG1	1.841	2.444	-0.618	0.000	-0.618	0.000
KBG2	0.000	0.000	0.000	0.000	0.000	0.000
KBG3	0.126	2.529	-0.297	2.116	-0.776	1.648
KBG7	0.000	0.000	0.000	0.000	0.000	0.000
KBG8	-0.482	2.130	-0.722	1.896	-0.343	2.265
KBG9	0.808	2.091	-0.387	0.911	-1.311	0.000
G-Av.	0.382	1.532	-0.338	0.821	-0.508	0.652

Table 6.4: Behavior of the three compared methods on instances of Set1.b

- Second, CM has a better behavior that the second version of B1.16 (with τ=1). Indeed, it improves three solutions (instances) among the sixth ones, where CM's percentage improvement varies from 0.9113% (instance KP9) to 2.11561% (instance KPG3). Moreover, both CM and B1.16 match the optimal densities for the rest of instances, i.e., KBG1, KBG2 and KBG7.

- Third, CM performs better than TSLA for the instances of Set1b. Indeed, CM provides a better result for two instances (KPG3 and KBG8) and it matches the

rest of the bounds achieved by TSLA. By excluding the solutions matched by CM, the percentage improvement varies from 1.64838% (KPG3) and 2.265% (KPG8).

- Furthermore, over the six instances, CM is able to reach two new solutions when compared to the best published bounds and matches the rest of the solutions (4).

	Cooperative Method's cpu		
Inst.	min	mean	max
KBG1	50.67	589.505	54.42
KBG2	0	0	0
KBG3	3.87	522.252	1934.46
KBG7	0	0	0
KBG8	57.01	1322.926	113.75
KBG9	128.34	2176.186	3476.85
G-Av.	39.98	685.14	929.91

Table 6.5: Variation of CM's runtime on instances of Set1b

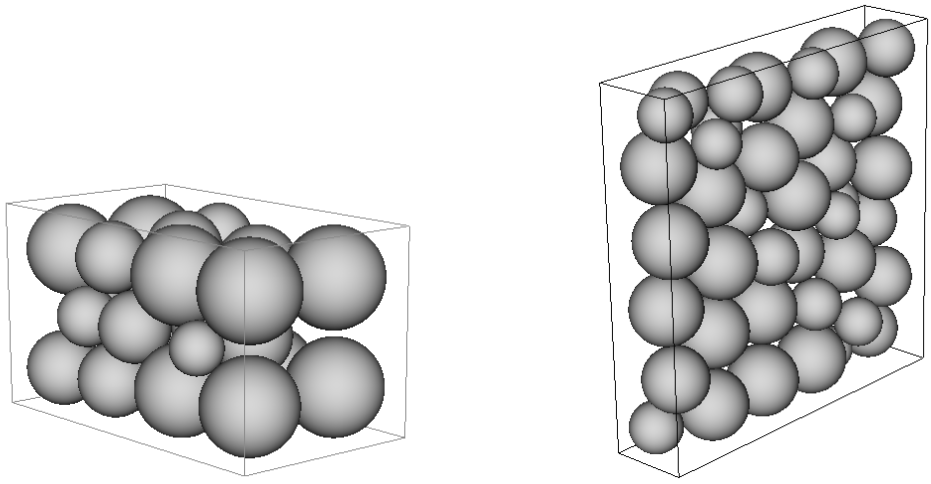
Moreover, as we have already mentioned that the runtime limit was fixed to one hour for all compared methods. For instances of Set1b, we also saved the best (average) runtime for which CM provides the best (average) bound, as shown in Table 6.5. Indeed, over the ten trials, and by excluding both KBG2 and KBG7 (for which the optimality is proven), one can observe that the minimum and maximum global average runtimes (last line, column 2 and column 4, respectively) is equal to 39.98 seconds and 929.91 seconds, respectively. Finally, the global average runtime over the ten trials is equal to 685.14 seconds, which is much less important than the one-hour considered as one of the stopping criteria.

We also provide the structure related to two instances of Set1b as illustrated in Figure 6.9 (KBG3 and KBG8 from the left-hand to the right-hand), where CM is able to improve their bounds (we recall that for the rest of instances, all methods are able to provide the optimal solutions, where all spheres are packed). These structures represent new solutions, where $D_{CM}(\text{KBG3}) = 53.507\%$ and $D_{CM}(\text{KBG8}) = 47.416\%$.

6.3.3 Behavior of CM on instances of Set2

In order to study the behavior of the proposed method on more hardness instances, we evaluated its performance on a set (noted Set2) containing 96 instances taken from Kubach *et al.* [45]. We then compared its obtained results to those reached by Kubach *et al.*'s parallel heuristic (noted B1.16).

Set2 is divided into four subsets following the number of spheres to pack: Set2a, Set2b, Set2c and Set2d, where each subset contains 24 instances with 20, 30, 40 and 50 spheres to pack, respectively. Set2a includes the instances noted from KP1 to KP24, Set2b is represented by the instances noted from KP74 to KP96, Set2c includes the instances noted from KP145 to KP240 and, Set2d contains the rest of the instances noted from



(a) KBG3 (b) KBG8

Figure 6.9: Illustration of the solutions' structures of two instances ((a) KBG3 and (b) KBG8) belonging to Set1b.

KP217 to KP240, respectively. Moreover, Kubach *et al.*'s [45] parallel heuristic has been run without runtime limit whereas CM's runtime limit was fixed to 7200 seconds, where the last bounds achieved by CM was considered as the final solution value (density) for the method. Because Kubach *et al.*'s [45] parallel heuristic uses a parameter τ, which serves to reduce the number of positions to attempt, we then considered both versions: the first one runs with the parameter $\tau = 0.8$ and the second one that considers all positions, i.e., with the parameter $\tau = 1.0$.

	Parallel B1.16 ($\tau = 0.8$)					
Subset	D_f	t_f	t_{succ}	D_{tot}	t_{tot}	nb_{opt}
Set2a	46.368	16	4	41.798	13	7
Set2b	49.224	550	0.01	46.519	458	4
Set2c	50.553	9438	0.01	49.439	8259	3
Set2d	51.042	8273	759	51.020	7647	2
G-Av.	49.29675	4569.25	190.755	47.194	4094.25	4
	Parallel B1.16 ($\tau = 1.0$)					
Subset	D_f	t_f	t_{succ}	D_{tot}	t_{tot}	nb_{opt}
Set2a	49.517	2208	6	44.067	1566	7
Set2b	50.901	53301	10646	48.941	35528	10
Set2c	51.148	469900	12978	51.565	222400	13
Set2d	51.605	405077	1250	52.950	203163	12
G-Av.	50.79275	232621.5	6220	49.381	115664.25	10.5

Table 6.6 shows the results realized by both versions of B1.16 (displayed in the first two-blocks of the table) and those obtained by the proposed cooperative method CM (the third block of the table). First, for all blocks, column 1 displays the data labels of the four

	Maximum and average results on ten trials: Cooperative Method										
	D_f		t_f		t_{succ}		D_{tot}		t_{tot}		nb_{opt}
Subset	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	
Set2a	48.111	**49.632**	250.75	638.30	15.86	48.09	42.949	**44.111**	182.24	466.15	7
Set2b	49.485	**50.922**	1330.86	3062.74	274.27	595.23	47.917	**48.954**	890.61	2034.61	10
Set2c	49.762	**51.234**	1838.34	2542.66	398.34	767.27	50.853	**51.604**	1058.34	1580.99	13
Set2d	46.117	**51.615**	3603.63	5176.80	159.14	197.46	48.021	**52.955**	1881.39	2687.13	12
G-Av.	48.369	**50.851**	1755.89	2855.12	211.91	402.01	47.435	**49.406**	1003.15	1692.22	10.50

Table 6.6: Behavior of both versions of B1.16 and CM on the instances of Set2.

subsets of Set2. Second, D_f denotes the average density of each subset of Set2 (Set2a, ..., Set2d) whenever the method is not able to pack all spheres and t_f is the average final runtime that needs the method to achieve the final solutions (D_f). Third, t_{succ} is the average final runtime that needs each method to pack all spheres (with success), D_{tot} (resp. t_{tot}) represents the average density (resp. its corresponding average runtime) for all instances of each subset. Finally, nb_{opt} is the number of instances solved to optimality by the corresponding method. In addition, because CM considers ten trials, we then display (in the third block) the average (noted Mean) and the maximum (noted Max) values of D_f, t_f, t_{succ}, D_{tot}, t_{tot} and nb_{opt}, respectively.

				Cooperative Method					
Set2a ($n = 20$)				Average			Max		
Inst	L	H	W	D	#d	t_s	D	#d	t_s
KP1	10	30.345	5	46.4265	15.6	64.89	47.670	17	33.53
KP2	10	20.23	7.5	46.967	17	107.50	47.687	17	387.70
KP3	10	15.172	10	50.4692	18	940.59	51.509	18	1269.20
KP4	10	10.416	5	44.1237	15.9	202.81	47.336	17	185.78
KP5	10	6.944	7.5	45.4693	15.7	76.74	49.249	18	184.18
KP6	10	5.208	10	45.5844	16.4	282.51	47.382	17	588.84
KP7	10	13.146	5	46.1502	16.6	341.61	47.597	17	623.24
KP8	10	8.764	7.5	49.6914	17.5	188.56	51.630	19	829.53
KP9	10	6.573	10	50.438	18	449.03	51.175	18	902.98
KP10	10	7.171	5	48.4374	18.2	115.91	50.179	19	131.60
KP11	10	4.781	7.5	47.1886	17.6	105.62	48.384	18	345.56
KP12	10	3.586	10	47.4227	16.9	209.15	48.684	17	611.21
KP13⋆	10	1.97	5	43.8306	18.6	110.88	45.860	20	336.49
KP14⋆	10	1.97	7.5	30.573	20	0.00	30.573	20	0.00
KP15⋆	10	1.97	10	22.93	20	0.00	22.930	20	0.00
KP16	10	11.916	5	48.4367	17.9	70.94	49.260	18	37.87
KP17	10	7.944	7.5	52.4241	18.5	218.69	53.994	19	1187.47
KP18	10	5.958	10	48.7011	18.8	126.03	49.916	19	209.76
KP19	10	5.136	5	51.8485	18.6	635.36	53.089	19	3251.79
KP20	10	3.424	7.5	48.1165	17.7	126.75	49.011	18	70.80
KP21⋆	10	3.29	10	42.934	20	0.10	42.934	20	0.10
KP22⋆	10	1.956	5	33.511	20	0.01	33.511	20	0.01
KP23⋆	10	1.956	7.5	22.34	20	0.02	22.340	20	0.02
KP24⋆	10	1.956	10	16.755	20	0.03	16.755	20	0.03
G-Av.				42.9487	18.06	182.24	44.1106	18.54	466.15

Table 6.7: Average and best densities (with their runtimes) reached by CM over the ten trials for the instances of Set2a

From Table 6.6, we can remark what follows.

- CM outperforms both parallel versions of B1.16. Indeed, CM's global average density is equal to 49.406% whereas the first (resp. second) version of the parallel B1.16 is equal to 47.194% (resp. 49.381%).
- CM has a better behavior when comparing its obtained results to those reached by the first version of parallel B1.16. Indeed, the percentage gap improvement of CM (CM−B1.16)×100 (between both average densities) varies from 0.5% (instances of Set2d) to 4.4% (instances of Set2a). The percentage related to filling spheres into the container increases according to the number of spheres to pack, even if the gap related to the improvement doesn't have the same behavior (for example, see the average densities for the instances of Set2b and Set2c).
- CM's results remain better than those reached by the second version of parallel B1.16. Indeed, its gap remains positive, where its average density varies from 41.798% to 51.02% whereas CM's average density increases for realizing the value of 44.111% to 52.955%, respectively.
- Even if the average runtime of the first version of B1.16 remains interesting (4094.25 seconds), that of CM is very impressive (1884.61 seconds). Indeed, both average runtimes show that CM is 2.17 times faster than the first parall version of B1.16.
- As pointed above, the second parallel version of B1.16 has a great chance to find solutions with hight quality, because all eligible positions of each candidate sphere to pack are considered. Despite its unlimited runtime, CM is able to achieve the same number of optimal solutions (at the same time it improves the global average density of the instances of Set2, as discussed above). In fact, CM's average runtime is now 61.37 times faster than that used by the second parallel version of B1.16 for achieving the average densities exposed in the second block of the table (even if the processors used by each method are slightly different).

Second and last, Tables from 6.7 to 6.10 report the average results realized by CM for the four subsets of instances (Set2a, Set2b, Set2c and Set2d, respectively) over the ten trials. Column 1 (of each table) displays the label of the instance and reports its data information in the three next columns. Column 5 (resp. column 6) reports CM's average density (resp. average runtime) for the ten trials whereas column 7 (resp. column 8) shows the best density (resp. runtime) achieved (resp. needed) by CM over the ten trials.

From Tables 6.7 to Table 6.10, one can observe that CM confirms its superiority by achieving several optimal solutions (as shown in column 10 of Table 6.6: an average of 10.5 optimal solutions realized by CM, for which all spheres were packed) almost of the four ones realized by the first parallel version of B1.16 (cf. Tables from 6.7 to 6.10, all instances marked with symbol "$\star$"). On the other hand, even CM has a better behavior than that the second parallel version of B1.16, it realizes tha same number of optimal solutions (marked with the symbol "$\star$").

Set2b ($n = 30$)				Cooperative Method					
				Average			Max		
Inst	L	H	W	D	#d	t_s	D	#d	t_s
KP73	10	35.57	5	47.3891	24.3	387.82	48.112	26	605.28
KP74	10	23.714	7.5	50.9428	26.7	652.83	51.957	28	250.51
KP75	10	17.785	10	52.595	27.9	1954.72	53.476	28	4413.77
KP76	10	14.506	5	47.0779	24.4	1858.68	48.948	28	5421.99
KP77	10	9.671	7.5	50.0356	25.9	682.22	52.037	26	1818.54
KP78	10	7.253	10	49.1036	25.9	1003.15	51.693	28	2026.08
KP79	10	25.011	5	48.23256	27.1	1664.00	49.58	28	3470.98
KP80*	10	16.674	7.5	52.922	29	1596.04	55.001	30	3902.78
KP81	10	12.506	10	52.94614	35.3	3461.10	54.0093	29	7110.85
KP82	10	9.167	5	53.042	28.8	898.44	53.69	29	344.71
KP83*	10	6.111	7.5	53.651	28.6	306.21	55.001	30	902.78
KP84	10	4.583	10	49.3988	27.3	1381.51	52.928	28	5398.43
KP85	10	2.176	5	43.6295	21.3	698.28	44.267	23	1023.87
KP86*	10	1.99	7.5	40.094	30	0.00	40.094	30	0.00
KP87*	10	1.99	10	30.071	30	1.49	30.071	30	1.49
KP88	10	22.206	5	48.3313	26.6	1758.30	49.608	28	4950.59
KP89*	10	14.804	7.5	54.6358	29.7	43.38	55.001	30	50.38
KP90*	10	11.103	10	55.001	30	17.70	55.001	30	17.70
KP91*	10	9.019	5	54.0305	29.3	363.60	54.997	30	662.84
KP92*	10	6.012	7.5	55.003	30	414.00	55.003	30	414.00
KP93	10	4.509	10	50.9798	27.9	1582.21	52.981	29	5154.99
KP94	10	1.926	5	49.0881	25.4	648.73	49.624	27	887.83
KP95*	10	1.926	7.5	35.322	30	0.00	35.322	30	0.00
KP96*	10	1.926	10	26.492	30	0.32	26.492	30	0.32
Av.				47.9173	27.98	890.61	48.9539	28.54	2034.61

Table 6.8: Average and best densities (with their runtimes) reached by CM over the ten trials for the instances of Set2b

Set2c ($n = 40$)				Cooperative Method					
				Average			Max		
Inst	L	H	W	D	#d	t_s	D	#d	t_s
KP145	10	44.958	5	47.2917	31.3	1346.11	47.974	32	2692.12
KP146	10	29.972	7.5	52.7601	34.1	3121.53	53.496	37	4213.43
KP147	10	22.479	10	53.821	35.5	2379.24	54.439	39	3397.87
KP148	10	20.267	5	45.3572	29.8	1490.51	46.597	39	3397.87
KP149	10	13.511	7.5	51.2145	35.9	1389.44	53.989	37	1664.17
KP150	10	10.134	10	50.6889	32.1	1418.10	53.805	37	422.23
KP151	10	27.612	5	49.2829	35.7	2666.36	49.874	36	3697.03
KP152*	10	18.408	7.5	54.999	40	0.10	54.999	40	0.10
KP153*	10	13.806	10	54.999	40	19.12	54.999	40	19.12
KP154*	10	12.816	5	53.9366	37.3	2038.65	55.001	40	5415.09
KP155*	10	8.544	7.5	55.001	40	13.80	55.001	40	13.80
KP156*	10	6.408	10	55.001	40	84.28	55.001	40	84.28
KP157	10	3.316	5	52.05369	35.9	1487.85	53.7899	38	1264.11
KP158*	10	2.211	7.5	45.5288	29.7	1092.41	46.752	31	1087.51
KP159*	10	1.968	10	46.333	40	280.84	46.333	40	280.84
KP160*	10	25.748	5	50.6741	36.5	1834.21	53.001	38	1549.36
KP161*	10	17.165	7.5	54.831	39.7	928.99	55.00	40	1140.20
KP162*	10	12.874	10	54.999	40	29.74	54.999	40	29.74
KP163*	10	9.655	5	54.998	40	9.16	54.998	40	9.16
KP164*	10	6.436	7.5	55.004	40	23.01	55.004	40	23.01
KP165*	10	4.827	10	54.3789	39.6	1750.22	55.00	40	2958.68
KP166	10	2.433	5	48.7117	35	1996.03	49.854	37	4583.57
KP167*	10	1.986	7.5	44.914	40	0.00	44.914	40	0.00
KP168*	10	1.986	10	33.686	40	0.55	33.686	40	0.55
Av.				50.8527	37.00	1058.34	51.6044	38.38	1580.99

Table 6.9: Average and best densities (with their runtimes) reached by CM over the ten trials for the instances of Set2c

Set2d ($n = 50$)				Cooperative Method					
				Average			Max		
Inst	L	H	W	D	#d	t_s	D	#d	t_s
KP217	10	55.93	5	47.3276	40.7	3004.12	47.887	41	3260.51
KP218	10	37.287	7.5	52.6881	48.3	3932.58	53.252	48	5432.78
KP219	10	27.965	10	53.5656	48.8	4770.90	54.358	49	6589.09
KP220	10	23.291	5	47.9998	43.9	3586.39	48.993	45	6973.36
KP221	10	15.527	7.5	52.6161	47.6	5024.99	53.223	48	6303.38
KP222	10	11.645	10	52.2282	46.9	4387.99	52.705	48	6997.95
KP223	10	34.766	5	50.2251	46.6	3200.08	51.668	47	2754.82
KP224*	10	23.177	7.5	55.001	50	0.00	55.001	50	0.00
KP225*	10	17.383	10	55	50	27.78	55.000	50	27.78
KP226*	10	15.043	5	55.002	50	46.05	55.002	50	46.05
KP227*	10	10.029	7.5	55	50	42.02	55.000	50	42.02
KP228*	10	7.522	10	54.722	49.6	555.16	54.998	50	660.27
KP229	10	4.414	5	53.6287	48.7	3168.16	54.417	49	4899.55
KP230	10	2.943	7.5	51.1253	47	3249.02	51.866	48	7126.74
KP231	10	2.207	10	47.1602	41.2	2126.06	47.38	42	1851.69
KP232	10	27.42	5	51.9017	48.1	2229.54	52.5	49	3404.48
KP233*	10	18.28	7.5	55	50	0.00	55.000	50	0.00
KP234*	10	13.71	10	55	50	0.00	55.000	50	0.00
KP235*	10	11.337	5	55.001	50	63.41	55.001	50	63.41
KP236*	10	7.558	7.5	55.001	50	0.00	55.001	50	0.00
KP237*	10	5.669	10	54.996	50	12.20	54.996	50	12.20
KP238*	10	3.334	5	54.7558	49.5	1163.09	55.004	50	1517.80
KP239	10	2.223	7.5	49.7962	45.6	4563.74	51.131	47	6527.19
KP240*	10	1.97	10	46.544	50	0.00	46.544	50	0.00
Av%				52.5536	48.02	1881.39	52.9553	48.38	2687.13

Table 6.10: Average and best densities (with their runtimes) reached by CM over the ten trials for the instances of Set2d

6.4 Conclusion

In this chapter the three-dimensional sphere packing problem is solved by using a cooperative method that combines three main features: a best-local position procedure, an intensification stage and a diversification stage. The first procedure ensures a starting feasible solution using a basic greedy local strategy. The second stage tries to solve a series of decision problems in order to pack a subset of complementary spheres. The third stage forces the current solution to remove some packed items and then replaced them with other promising spheres. The performance of the proposed method is evaluated on benchmark instances taken from the literature where the achieved results are compared to those reached by some recent methods of the literature. The proposed method remains competitive and succeeded in yielding new solutions for many instances.

Chapter 7

Particle swarm optimization-based approach for identiqual sphere packing problem

Contents

In this chapter, we propose to solve packing identical spheres with a particle swarm optimization-based algorithm. Given a set of identical spheres and a container (open or spherical), the aim of the problem is to find a smallest container that contains all spheres without overlapping between spheres and between spheres and the container. The particle swarm optimization cooperates with an efficient continuous local optimization that serves either to repair the non-feasibility of solutions or improve their quality. The performance of the proposed algorithm is evaluated on a set of standard benchmark instances and its obtained results are compared to those achieved by the more recent published methods available in the literature. Encouraging results have been obtained.

7.1 Introduction

This chapter deals with solving the Identical Sphere Packing (noted ISP) problem into two types of containers: the *Open container* (noted O-ISP) with unlimited length and the *Spherical container* (noted S-ISP) with unlimited radius. An instance of both problems is characterized by a set N of n identical spheres, where each sphere $i \in N = \{1, \ldots, n\}$ is represented by its radius $r_i = 1$ and the container is either $\mathcal{P}$ with fixed width W and height H but unlimited length $L = \infty$ or $\mathcal{S}$ of unlimited radius $R_0 = \infty$. For both versions of the problem, the goal is to minimize the length L (if the container $\mathcal{P}$ is open) or the radius R_0 (if the container $\mathcal{S}$ is spherical) such that all items of N are positioned in $\mathcal{P}$ or R_0, without overlapping between spheres and between spheres and the container.

Both version O-ISP and S-ISP can be encountered in several real-life applications, like automated radio-surgical treatment planning where the problem is used as a tool for solving the radio surgical treatment planning (cf. Sutou and Dai [69]) and, materials science where a random sphere packing problem is used as model for studying the dynamic behavior of granular material systems (cf. Li and Ji [51]).

On the one hand, O-ISP can be formulated as follows:

$$\text{Minimize} \quad L \tag{7.1}$$
$$d(i,j) \geq (r_i + r_j), \ \forall (i,j) \in N^2, \ i < j \tag{7.2}$$
$$r_i \leq x_i \leq L - r_i, \forall i \in N \tag{7.3}$$
$$r_i \leq y_i \leq H - r_i, \forall i \in N \tag{7.4}$$
$$r_i \leq z_i \leq W - r_i, \forall i \in N \tag{7.5}$$

where $d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}, i \in N, \ j \in J, i \neq j$. Eq. (7.1) represents the objective function that minimizes the length L of the target container $\mathcal{P}$. Eq. (7.2) represents the quadratic constraints that ensure the non overlapping between any pair of distinct spheres. Eqs. from 7.3 to 7.4 denotes the linear constraints that represent the sphere's feasibility when positioned into the container $\mathcal{P}$ of length L.

On the other hand, the problem S-ISP can be stated as follows:

$$\text{Minimize} \quad R_0 \tag{7.6}$$
$$d(i,j) \geq (r_i + r_j), \ \forall (i,j) \in N^2, \ i < j \tag{7.7}$$
$$d(0,i) \leq (R_0 - r_i), \forall i \in N \tag{7.8}$$

where $d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}, i \in N, \ j \in J, i \neq j$ and $d(i,0) = \sqrt{x_i^2 + y_i^2 + z_i^2}, i \in N$. Eq. (7.6) is the objective function that minimizes the radius R_0 of the target container $\mathcal{S}$. Eq. (7.7) represent the quadratic constraints that ensure the non-overlapping between any pair of distinct spheres and Eq. (7.8) denote the quadratic constraints representing the sphere's feasibility when positioned into the container of radius R_0.

For both O-ISP and S-ISP, a packing can be represented by a vector

$$\overrightarrow{X} = (F, x_i, y_i, z_i, ..., x_n, y_n, z_n),$$

where F, $F = L$ or R_0, denotes the length or radius of the target container $\mathcal{P}$ or $\mathcal{S}$ and $(x_i, y_i, z_i, ..., x_n, y_n, z_n)$ is the coordinates of positions of the n packed spheres.

In order to measure the (non)feasibility of a given packing $\overrightarrow{X}$, the amount of overlapping is measured following quantities represented in Table 7.1.

Between	O-ISP	S-ISP
sphere and container	$O_{i,x} = \max\left\{0, r_i + \lvert x_i\rvert - \frac{1}{2}L\right\}$ $O_{i,y} = \max\left\{0, r_i + \lvert y_i\rvert - \frac{1}{2}H\right\}$ $O_{i,z} = \max\left\{0, r_i + \lvert z_i\rvert - \frac{1}{2}W\right\}$	$O_{0,i} = \max\{0, d(i,0) + r_i - R\}$
sphere and sphere	$O_{i,j} = \max\{0, r_i + rj - d(i,j)\}$	$O_{i,j} = \max\{0, r_i + rj - d(i,j)\}$

Table 7.1: Function's overlapping for both problems: between the positioned spheres and between spheres and the container

Hence, according to both termes of overlapping, following the couple of spheres or a sphere with the container, the first overlap $E(\overrightarrow{X})$ associated to the first version of the problem O-ISP is given as follows:

$$E(\overrightarrow{X}) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} O_{i,j}^2 + \sum_{i=1}^{N} (O_{i,x} + O_{i,y} + O_{i,z})$$

and that associated to the second problem S-ISP is given as follows:

$$E(\overrightarrow{X}) = \sum_{i=0}^{N-1} \sum_{j=i+1}^{N} O_{i,j}^2.$$

The remainder of the chapter is organized as follows. Section 7.2 describes the basic steps of the particle swarm optimization and its adaptation for solving both versions of the packing problem. Section 7.2.2 discusses the main principle of the proposed method and the cooperation used for either repairing non-feasibility of the solutions or the improvement of the quality of the solutions at hand. Section 7.3 evaluates the performance of the proposed method on benchmark instances, where its achieved results are compared to those reached by recent algorithms available in the literature. Finally, Section 7.4 concludes by summarizing the contribution of the paper.

7.2 A particle swarm optimization-based algorithm for the identical sphere packing

7.2.1 Particle swarm optimization-based algorithm

PSO is a meta-heuristic that is based on a population (cf. Kennedy *et al.* [42]), where it can efficiently explore several spaces of candidate solutions during the search process. Like other meta-heuristics, PSO does not guarantee the optimality of the solutions achieved, but it generally ensures an experimental convergence to solutions of hight-quality. Note also that the method is also interesting regarding its simplicity and the number of parameters that are used in contrast to other evolutionary methods.

Each particle in PSO characterizes a solution of the given problem, where each of them is represented by a vector $\vec{X}$ of positions on the space. The best solution visited is represented by the best position (noted p_{Best}) with its velocity $\vec{v}$. Furthermore, each particle shares informations with other particles of the population (swarm) and updates its actual positions by using a simple formula which acts on the velocity; that is defined as follows:

$$v_i^t = \omega \times v_i^{t-1} + c_1 \times \upsilon \times \left[p_{Best} - x_i^{t-1}\right] + c_2 \times \nu \times \left[g_{Best} - x_i^{t-1}\right] \tag{7.9}$$

and

$$x_i^t = x_i^{t-1} + v_i^t, \tag{7.10}$$

where Eq. (7.9) acts on the velocity and Eq. (7.10) acts on the particle's positions.

One can observe that Eq. ((7.9)) is composed of three parts:

(i) $\omega \times v_i^{t-1}$: it represents the i-th particle's velocity at iteration $(t-1)$, where ω denotes the inertia weight (introduced by Shi *et al.* [64]). Such a weight tries to control the magnitude of the "old velocity" and usually takes its values in the interval $[0.4, 0.9]$.

(ii) $\left[p_{Best} - x_i^{t-1}\right]$: it represents a natural tendency of a particle to return to its best position, namely p_{Best}.

(iii) $\left[g_{Best} - x_i^{t-1}\right]$: it represents the tendency of a particle to follow the best position, namely g_{Best}, reached by any member belonging to its neighborhood.

In the global version of PSO, the neighborhood consists of the whole population and g_{Best} represents the best position. In contrast, if the neighborhood is a subset of a population, then the best position is called "local best" position (noted l_{Best} – cf. Parsopoulos *et al.* [63]). Finally, the other parameters, i.e., c_1 and c_2, represent the cognitive and social factors, respectively, where $c_1 + c_2 \leq 4$, whereas both υ and ν are randomly generated in the interval $[0, 1]$; these values are used to determine the degree of influence of p_{Best} and

g_{Best}, respectivelty. Of course, p_{Best} and g_{Best} associated to each particle are evaluated according to the objective function.

A new version of PSO, called COnstriction Coefficient PSO (noted COPSO), has been proposed by Clerc [14] and Clerc and Kennedy [15], where the goal of such a version is to ensure both convergence and stability of PSO. In this case, the velocity is updated by using a different way:

$$v_i^t = K \times \left(v_i^{t-1} + c_1 \times \upsilon \times \left[pBest - x_i^{t-1} \right] + c_2 \times \nu \times \left[gBest - x_i^{t-1} \right] \right) \tag{7.11}$$

and

$$x_i^t = x_i^{t-1} + v_i^t, \tag{7.12}$$

where K denotes the restriction factor which is defined according to c_1 and c_2:

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4 \times \varphi}|} \tag{7.13}$$

such that $\varphi = c_1 + c_2$ (of Eq. (7.11) for evaluating x_i^t of Eq. (7.12)) and $\varphi > 4$ (of Eq. (7.13)). Experimentally, φ is generally setting equal to the value 4.1, where $c_1 = c_2 = 2.05$ and $K = 0.729$ (as confirmed in our experimental part for the both versions of the sphere packing problem).

Because COPSO has a better performance than that of PSO, we then considered an adaptation of COPSO for solving both Open ISP (noted O-ISP) and Spherical ISP (noted S-ISP).

The main steps of the standard COPSO are given in Algorithm 19. It starts with a population of size K. The initialization step (line 2) searches for the position $\vec{X}^i$ of each particle i which are randomly generated or by using a simple heuristic. The best position with value p_{Best} is setting equal to the best solution of the starting population (line 3) and p_{Best}^i of each particle i is setting equal to the value of the current position $\vec{X}^i$ and the velocity vector is setting equal to zero. Therefore, the fitness of each particle is evaluated, where the best fitness reached is stored in g_{Best} (lines from 4 to 12). Furthere, both velocity and position of each particle i are updated according to equations (7.11) and (7.12) above (lines 14 and 15). At the end of the algorithm, it exits with best position visited that is considered as the best solution of the problem.

7.2.2 PSO and the sphere packing problem

The three-dimensional identical sphere packing problem is scientifically challenging, because it can be modeled as a nonlinear programming where both continuous positions of the spheres and combinatorial optimization aspect of these positions can be integrated to

Algorithm 19 – A standard version of the algorithm

Input. A population of size K and the different parameters used.

Output. A best particle of value g_{Best}.

Initialization Step.
Generate the position $\vec{X}^i$ of each particle.
Set $p^i_{Best} = \vec{X}^i$ and let $\vec{V}^i$ be the velocity vector.
Iterative step
while (the runtime limit is not performed) **do**
    for (each particle $i \in K$) **do**
        Compute its fitness function $F_{\vec{X}^i}$
        if $(F_{\vec{X}^i} < F_{\vec{p}_{Best}{}^i})$ **then**
            $p_{Best} = \vec{X}^i$
        end if
    end for
    Choose the particle realizing the best value $F_{\vec{p}_{Best}{}^i}$ as g_{Best}
    for (each particle $i \in K$) **do**
        Compute the particle's velocity according to Eq. (7.11)
        Update the particle's position according to Eq. (7.12)
    end for
end while
Return the best solution achieving the best global solution value g_{Best}.

its resolution. In this case, several local optimum, with the same value but with different configurations, can be obtained which makes the problem more complex to solve.

Because the problem contains a set of diversified solutions, we then propose to approximately solve it by using a three-steps-based heuristic. The proposed method is based upon COPSO using the following stages:

(i) Creating a starting population of feasible configurations: it can be obtained by adapting the truncated tree search procedure proposed by Hifi and Yousef ([38, 36]). Indeed, one can use the quick version of such a method in order to reach the starting population containing K feasible particles. Each particle is represented by a vector $\vec{X}$ (as described above).

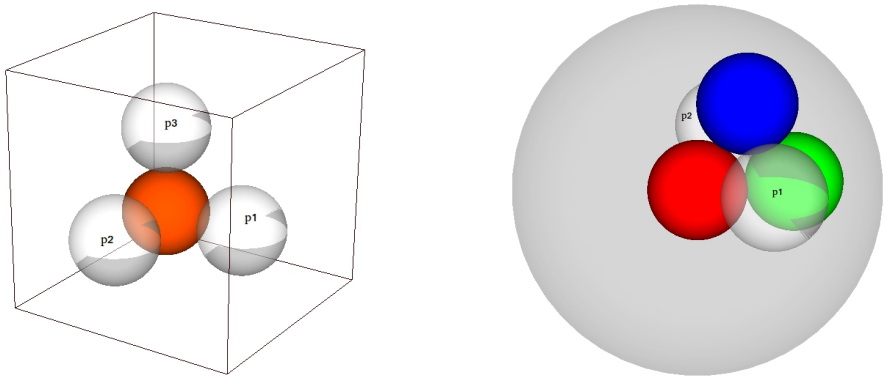
(ii) Applying PSO for solving the sphere packing problem.

(iii) At each iteration of PSO, a continuous local optimization is used for either repairing or improving the (quality of the) solutions.

In what follows, we first describe the procedure of generation the population of particles. Second, we present the continuous local optimization role. Third and last, the proposed particle swarm optimization-based algorithm is presented.

7.2.3 Generating the population of particles with a greedy procedure

The proposed method uses a population of K particles. Herein, a quick version of Hifi and Yousef's [38] algorithm is applied in order to generate the starting diversified population. The algorithm is considered as a truncated tree search, where at each level of the developed tree, only a subset of elite nodes are selected for further branching and the other ones are discarded, where no backtracking is performed. Moreover, the cardinality of the selected elite nodes is fixed to ω, where ω is a small nonnegative integer defined experimentally. We then adapt a quick version of the method, where a random position among all eligible positions is selected for positioning the current sphere into the container. Of course, such a process can be viewed as a Greedy Procedure (noted GP), which tries to generate quick random paths of the developed tree; that are a set of feasible solutions for the problem.



(a) O-ISP's initial node (b) S-ISP's initial node

Figure 7.1: Illustration of the first positioned item into the container.

A path, representing a feasible solution for either O-ISP or S-ISP, can be generated by iteratively positioning the items of N, one by one, at the (random) selected positions inside the container. At each step of the procedure, let consider the following notations:

- I_i: the set of items of N already packed in the current container $\mathcal{P}$ (or S).
- $\overline{I_i}$: the set of items of N which are not yet packed in the current container $\mathcal{P}$ (or S).
- P_{I_i}: the set of distinct eligible positions for the next item i to pack given the set of packed items I_i.

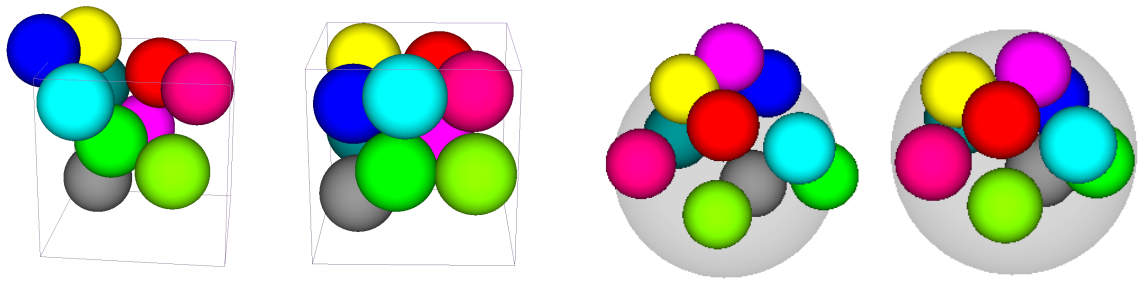
A position $p_{i+1} \in P_{I_i}$ (for the i-th item) is determined according to three elements, namely e_1, e_2 and e_3. In this case, an element is either an item of N already positioned (representing I_i) or, either one of the six faces of the open container P or the inner-side of the spherical container S. By using the aforementioned representation, Figure 7.1 illustrates the starting node of each path, where the first item (sphere) is positioned

at (r_i, r_i, r_i) for O-ISP and the first three items of S-ISP are successively positioned at $(0, 0, 0)$, $(2, 0, 0)$ and $(1, \sqrt{3}, 0)$, respectively.
Hence, (i) $I_i = \{1\}$ for O-ISP and $I_i = \{1, 2, 3\}$ for S-ISP, (ii) $\overline{I_i} = \{2, ...N-1\}$ for O-ISP and $\overline{I_i} = \{4, ...N-3\}$ for O-ISP and, (iii) $P_{I_i} = \{p_1, p_2, p_3, \ldots\}$ is the set of eligible positions for the next branching.

We recall that a population-based method is generally used for its diversifying aspect on the solutions, where each solution is built with different characteristics. For the proposed PSO, on the one hand, the above procedure is called K times for generating all initial particles of the population. On the other hand, the parameter ω was fixed to K in order to try a better diversification of particles.

7.2.4 A continuous local optimization

The proposed algorithm uses a special continuous local optimization which has been already proposed by Wang *et al.* [72] (called "Quasi-Physical" Procedure, noted QPP). The main principle of such a method can be summarized as follows (as illustrated in Figure 7.2): (i) generating a (un)feasible solution and (ii) repairing the current solution in order to making it feasible.



(a) Non-feasible packing (b) A repaired solution (c) Non-feasible packing (d) A repaired solution

Figure 7.2: Illustration of QPP's mechanism for both versions of the problem

In QPP, the container is considered as a “fixed elastic container" and all the spheres as “smooth elastic solids" which are forced to be included into the container. According to the elasticity mechanism, there exist some conjugated extrusion elastic force whenever some spheres are deformed by other ones or by the container. In this case, a series of complicated distorsions would occur due to these elastic forces. During the search process, the overlapping gap (either between the deformed spheres or that realized between spheres and the container) will be reduced to zero. Reducing such deformations to zero induces the converge of the procedure to a local optimum (for more details, the reader can refer to Wang *et al.* [**?**]).

7.2.5 An overview of the proposed algorithm for the packing problem

The main steps of the proposed method is detailed in Algorithm 20. First, it starts by initializing a starting feasible position for each particle; that is obtained by applying the truncated tree search procedure proposed in Hifi and Yousef ([38, 36]). Each of these positions is represented by a vector

$$\vec{X_i}^{\,k} = (F_i^k, x_1^k, y_1^k, z_1^k, x_2^k, y_2^k, z_2^k ... x_n^k, y_n^k, z_n^k)$$

and an initial velocity vector v_i^k randomly generated in the interval $(-1, 1)$.

Second, the best position of each particle is setting equal to the staring one (i.e., $p_{Best} = \vec{X_i}$). Further, the fitness function $f(\vec{X_i}^{\,t})$ associated to each particle i is setting equal either to L (for the first version of the problem) or R_0 (for the second one); that represents the minimum length of the target container $\mathcal{P}$ or the minimum radius of the target container $\mathcal{S}$, respectively.

Algorithm 20 . PSO-Based Algorithm (PSO-BA) for the sphere packing problem

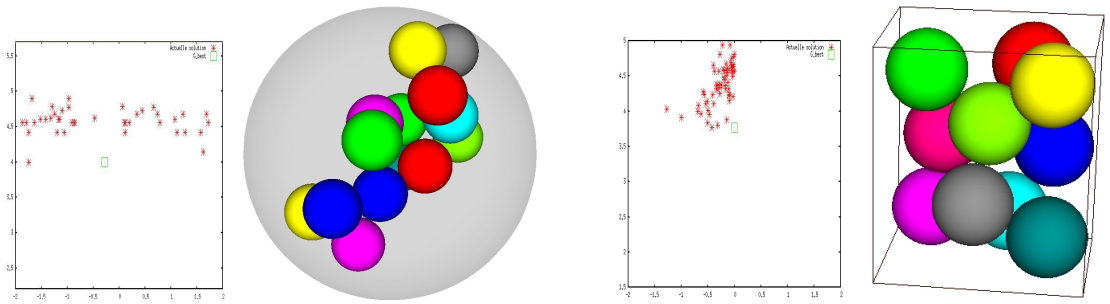
Input. A population of size K.

Output. A best particle realizing the value g_{Best}.

Initialization Step
Call the truncated tree search procedure and assign the resulting solution as its initial solution.
Set $p_{Best} = \vec{X_i}$ and its velocity to v_i^t.
Iterative step
while (the runtime limit is not performed) **do**
    for each particle **do**
        Compute the fitness function $f\left(\vec{X_i}^{\,t}\right)$.
        if $\left(f(\vec{X_i}^{\,t}) < f(\vec{p_{Best}}^{\,t})\right)$ **then**
            $\vec{p_{Best}}^{\,t} = \vec{X_i}^{\,t}$
        end if
    end for
    Let g_{Best} be the best particles' fitness (realizing the maximum value of $f(\vec{p}_{Best}^{\,t})$)
    for each particle **do**
        Compute v_i^k according to Equation (7.11)
        Update $\vec{X_i}^{\,t}$ according to Equation (7.12)
        Compute the particle's penalty according to Equation (7.1)
        if $(E(\vec{X_i}^{\,k}) > \epsilon)$ **then**
            Apply the continuous local optimization (cf. Section 7.2.4)
        end if
    end for
end while
return g_{Best}

PSO-BA is an iterative procedure composed of three loops. The outer loop tries to control the stopping criteria (represented by a standard runtime limit used when sphere packing problems are solved) whereas the two internal loops undertake the search. The first inner loop (lines from 6 to 11) evaluates the position of each particle and its best position visited. At the end of the aforementioned loop, a best position visited by all particles is stored; that is considered as the global best position (noted g_{Best}). The second internal loop (from 13 to 20) starts by updating both velocity and position of each particle. During the search process, if a particle is positioned out of the feasible region, the continuous local optimization (described in Section 7.2.4) is called in order either to repair or to improve the solution at hand. Finally, the algorithm stops either when the runtime limit is performed or the velocity of each particle is close to zero.

To illustrate how PSO-BA proceeds, we consider two instances extracted from the literature. The first one is extracted from Stoyan *et al.*'s [66] (noted instance SYS1 in the experimental part, Section 7.3), this instance contains 10 identical spheres and it represents the packing problem into an open container (O-ISP). Whereas, the second one is extracted from M'Hallah *et al.* [55](noted instance 15 in the experimental part), this instance contains 15 identical spheres and it represents the packing problem into a spherical container. Note, it is to be emphasized that the example was only designed to illustrate how the algorithm performs through the iterations considered.

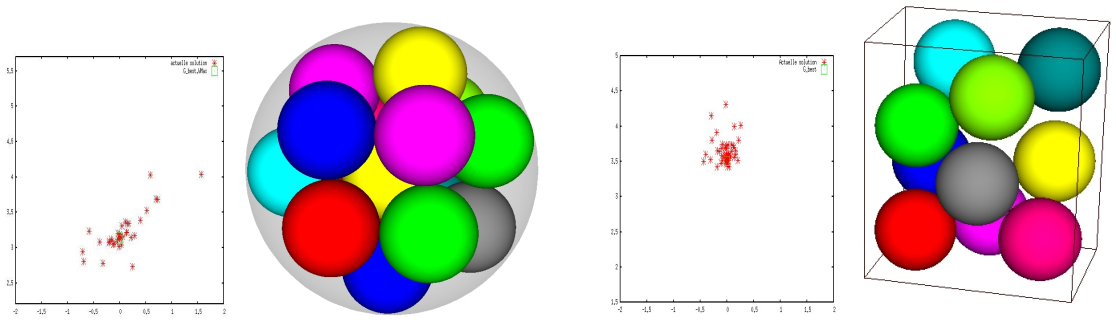


(a) Iteration 1. (b) $R_0 = 6.19806363587873$. (c) Iteration 1. (d) $L = 3.8995109762$.

Figure 7.3: Illustration of the starting solution achieved by GP: (i) on M'Hallah *et al.*'s instance containing 15 spheres for (a) and (b) and, (ii) on the instance SYS1 (figures (c) and (d)) with 10 spheres to pack.

First, the starting solutions that serve to initialize the population (its size is discussed in the experimental part, Section 7.3) are reached by applying the greedy procedure GP. In fact, GP is used as a random procedure for which each favorable position related to the current sphere to pack is randomly chosen among all eligible positions of that sphere. For M'Hallah *et al.*'s instance (resp. SYS1), the upper bound achieved at this stage is equal to $R_0 = 6.19806363587873$ (resp. $L = 3.8995109762$) as shown in Figure 7.3(a) which represents the initial population with its best partile structure in Figure 7.3(b) (resp. in

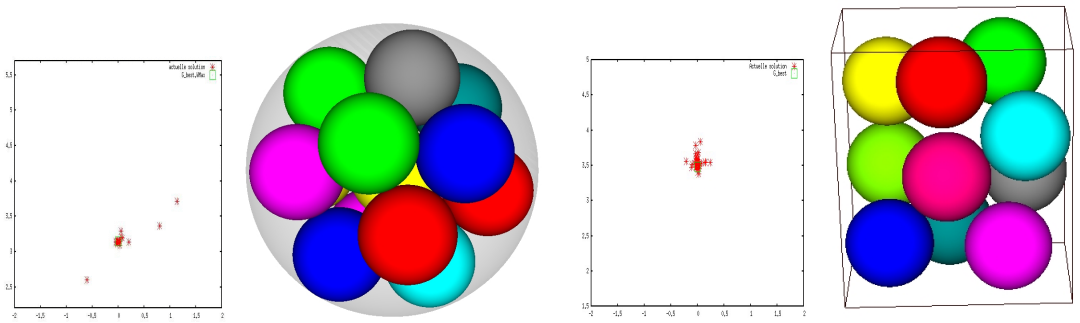
Figure 7.3(c) with its best particle structure in Figure 7.3(d)). These bounds are obtained at step 1 of PSO-BA.



(a) Iteration 500. (b) $R_0 = 3.1790353894595$. (c) Iteration 500. (d) $L = 3.4878305719$.

Figure 7.4: Illustration of the intermediate PSO-BA's solutions: (i) its upper bound on M'Hallah *et al.et al.*'s instance containing 15 spheres (figures (a) and (b) and, (ii) its upper bound on Stoyan *et al.*'s instance (for figures (c) and (d).

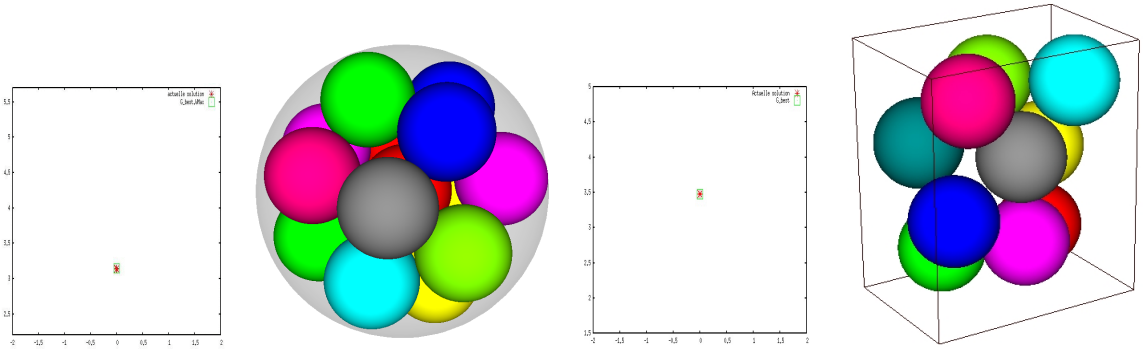
Indeed, after 500 iterations, PSO-BA improves the first bound thereby giving the value $R_0 = 3.1790353894595$ (cf. Figures 7.4(a) with its configuration in Figure 7.4(b)) and $L = 3.4878305719$ (cf. Figures 7.4(c) with its configuration in Figure 7.4(d)), respectively.



(a) Iteration 1000. (b) $R_0 = 3.14164270926649$. (c) Iteration 1000. (d) $L = 3.4765334775$.

Figure 7.5: Convergence of PSO-BA and illustration of the improved upper bounds of both M'Hallah *et al.*/Stoyan *et al.*'s instances containing 15 and 10 spheres, respectively.

Third, the additional rules are of interest whenever PSO is coupled with a continuous optimization procedure. Indeed, after 1000 iterations, one can observe that the upper bound decreases until reaching $R_0 = 3.14164270926649$ and $L = 3.4765334775$, respectively. The aforementioned upper bounds confirme that PSO combined with the continuous optimization is capable of finding good configurations when coupled with the random greedy procedure GP; it gives a set of diversified solutions for the method.



(a) Iteration 1550. (b) Final solution with $R_0 = 3.14164270926649$. (c) Iteration 2100. (d) Final solution with $L = 3.4718865883$.

Figure 7.6: Convergence of PSO-BA and illustration of the improved upper bounds of both M'Hallah *et al.*/Stoyan *et al.*'s instances containing 15 and 10 spheres, respectively.

Fourth, as the number of iterations increases, one can observe that PSO-BA's upper bounds decrease. Indeed, PSO-BA continues the search process by intensifying the research around the last solution and acting on repairing a series of obtained configurations until achieving the final solution (after 1550 iterations for the first problem and 2100 iterations for the second problem, respectively) with the bound $R_0 = 3.14164262494856$ (resp. $L = 3.471885145$); that are new upper bounds: (i) for M'Hallah *et al.*'s instance (Note that, for this instance, the best upper bound available in the literature, taken from M'Hallah *et al.* [55], is equal to 3.14164262494867) and (ii) for the instance SYS1 (the best upper bound available in the literature is extracted from Stoyan *et al.*'s [66] that is equal to $L = 3.4722$).

7.3 Computational results

This section evaluates the behavior of the proposed Particle Swarm Optimization-Based Algorithm (PSO-BA) on benchmark instances of the literature. For both versions of the problem, its achieved results are also compared to those reached by the best methods available in the literature. The proposed algorithm was coded in C++ and tested on an Intel Core 2 Duo (2.53 Ghz and with 4 Gb of RAM) and the runtime limit was fixed to two hour (that is considered as the standard runtime limit for solving such problems although most of the methods generally use a more larger limit).

7.3.1 The first version of the problem: O-ISP

For O-ISP, there exists two sets of instances (noted Set1 and Set2). The first set "Set1" contains seven instances (SYS1,...,SYS7) which are extracted from Stoyan *et al.* [66],

where the number of the spheres to pack varies from 10 to 150. The second set “Set2" contains thirty-eight instances taken from Birgin and Sobral [8]. For each instance, both dimensions W and H of the container are fixed to 9.5 whereas the number of the spheres to pack varies from 23 to 100.

Inst.	n	H	W
SYS1	10	5.9	5.1
SYS5	50	7.9	6.1
SYS7	150	9.9	7.1

Table 7.2: Characteristics of tested instances

7.3.1.1 Parametre settings

Since PSO-BA is a stochastic algorithm that uses a population of solutions, then each run can provide a different result. We then considered ten trials (runs) for each considered instance for both sets. On the other hand, before several tunings, we selected in what follows the values that lead better results with stability of the solutions, i.e., $c_1 = c_2 = 2.05$ (which are also used as the standard tunings on the standard version of the algorithm []).

In order to determine the population size K, PSO-BA was called with a population with different sizes, where K varies in the discrete intervalle $\{10; 20; 30; 40; 50; , 60; 70; 80\}$. Such a variation has been used for three special and standard instances of Stoyan *et al.* [66]: SYS1, SYS5 and SYS7 that represent small, medium and large-sized instances, respectively. As mentioned above, the runtime limit was fixed to 7200 seconds, which is generally considered as the standard runtime for such family of problems. Table 7.2 shows the characteristics of these instances whereas the best results obtained over the ten trials are reported in Table 7.3.

Inst	10	20	30	40
SYS1	3.5843552091	7.91515249	3.5674589738	3.5193542966
SYS5	8.6472459566	17.05213879	8.4604271787	8.362553048
SYS7	18.7628563652	11.44328249	16.6463912873	16.3971607426
Av	10.3314858436	12.1368579219	9.5580924799	9.4263560291
Inst	50	60	70	80
SYS1	3.4957892205	3.4805536456	3.4929224131	3.5095576343
SYS5	8.3366675383	8.3225393312	8.3283763787	8.4557590839
SYS7	16.4062977259	16.4179103644	16.4153490011	16.5092205817
Av	9.4129181616	**9.4070011137**	9.412215931	9.4915124333

Table 7.3: Behavior of PSO-BA when varying the size K of the population for the O-ISP

By analyzing the results of Table 7.3, one can observe that the best average results are realized globally for $K = 60$, which realizes an average bound of 9.4070011137. Two

other values of K are able to provide average values close to that achieved by the case previously mentioned, i.e., $K = 60$. Indeed, for $K = 50$ and $K = 70$, PSO-BA achieves an average bound of 9.4129181616 and 9.412215931, respectively. Because the accuracy of 10^{-9} is more relevant for these instances, for the rest of the experimental part, c_1 and c_2 are fixed to 2.05, the size K of the population is fixed to 60 and the runtime limit is fixed to 7200 second.

7.3.1.2 Behavior of PSO-BA on the first set of instances (Set1)

Herein, the behavior of PSO-BA is analyzed on the first set of instances Set1 and its performance is compared to that of Stoyan *et al.*'s [66] method (noted SYS). Both SYS and PSO-BA methods run with the same accuracy value for ϵ; that is setting equal to 10^{-9}.

All results realized by both methods PSO-BA and SYS are reported in Table 7.4. Columns from 1 to 4 show each instance information, column 5 displays SYS's length (noted L_{SYS}) (taken from Stoyan *et al.* [66]) whereas column 6 shows the runtime needed by the algorithm. Column 7 tallies the supper bound L_{PSO-BA} achieved by PSO-BA whereas column 8 displays the runtime needed for reaching the aforementioned bound. Finally, columns 9 reports the percentage improvement (when it happens) yielded by PSO-BA when compared to the results reached by SYS (in this case, the percentage improvement is computed as follows: $\%\text{Imp} = 100 \times \frac{L_{\text{SYS}} - L_{\text{APSO}}}{L_{\text{SYS}}}$).

				SYS		PSO-BA		
Inst	n	H	W	L_{STS}	t_{STS}	$L_{\text{PSO-BA}}$	$t_{\text{PSO-BA}}$	%Imp
SYS1	10	5.9	5.1	3.4722	8280	3.471885145	7112.60	0.00907
SYS2	20	7.9	6.1	3.5738	53550	3.459830895	6897.11	3.18902
SYS3	30	7.9	6.1	5.2251	3996	5.098522569	6976.23	2.42249
SYS4	40	7.9	6.1	6.7819	12564	6.714685502	6105.43	0.99109
SYS5	50	7.9	6.1	8.2759	26640	8.255066275	2912.89	0.25174
SYS6	100	9.9	7.1	11.0545	55044	11.032687899	6648.41	0.19731
SYS7	150	9.9	7.1	16.2638	105732	16.200758713	6865.31	0.38762
Av				7.8067	37972.29	7.747633857	6216.85	1.06405

Table 7.4: Performance of PSO-BA versus SYS on instances of Set1

The analysis of the results of Table 7.4 follows:

1. First, PSO-BA outperforms SYS since it is able to improve all the solutions reached by SYS. Indeed, on the one hand, PSO-BA's global average bound is equal to 7.7476338568 whereas SYS's average bound is equal to 7.806742. On the other hand, SYS needs 37972.28 seconds for achieving these bounds whereas PO-BA needs only an average value of 6216.85 seconds.

2. Second and last, when comparing PSO-BA's results to SYS's ones, one can observe

that the percentage of the improvement varies from 0.0090678734% (instance SYS1) to 3.1890174211% (instance SYS2). Globally, for the instances of Set1, the average percentage improvement is equal to 1.06405%.

We have already mentioned that ten trials of PSO-BA are considered for each instance of the set Set1. Table 7.5 reports the worst, best and average upper bounds achieved by PSO-BA for these ten trials and their corresponding runtimes (fixed to a maximum value of two hours).

	Particle Swarm Optimization-Based Algorithm					
Inst.	Min	t	Mean	t	Max	t
SYS1	3.471885145	7112.6	3.4805536456	3481.255	3.497834490	6395.63
SYS2	3.459830895	6897.11	3.5991936486	3617.482	3.807519076	874.57
SYS3	5.098522569	6976.23	5.2085338345	14680.006	5.344001557	2657.33
SYS4	6.714685502	6105.43	6.8432566543	4009.441	7.009642444	5137.61
SYS5	8.255066275	2912.89	8.3225393312	4866.997	8.536821504	5339.05
SYS6	11.032687899	6648.41	11.1138866023	4989.638	11.282461177	5513.64
SYS7	16.200758713	6865.31	16.4179103644	4839.634	16.662159977	2887.29
Av	7.74763385682	6216.85	7.85512486870	5783.49	8.02006288911	4115.02

Table 7.5: Behavior of PSO-BA on the ten trials for the first set of instances: Set1

7.3.1.3 Behavior of PSO-BA on instances of Set2

In this section, PSO-BA's behavior is analyzed on the second set of instances Set2. Its obtained results are also compared to those published by Birgin and Sobral [8] (also available on `https://www.ime.usp.br/~egbirgin/packing/packing`).

The results realized by both methods PSO-BA and BS are shown in Table 7.6. Columns from 1 to 4 report each instance information, column 5 shows Birgin and Sobral's upper bound (noted L_{BS}, where its was taken from Birgin and Sobral's [8]) whereas column 6 displays the runtime needed by that algorithm for realizing the length. Finally, column 7 shows the upper bound $L_{\text{PSO-BA}}$ achieved by PSO-BA whereas column 8 displays the runtime needed by that algorithm for reaching the given bound.

Note also that, on the one hand, Birgin and Sobral's (noted BS) performs with ϵ fixed to 10^{-4} whereas PSO-BA is applied with a more accuracy value that has been fixed to 10^{-9}. It means that PSO-BA uses a better accuracy and the overlapping is reduced to very tight values when compared to that realized with $\epsilon = 10^{-4}$; hence, the resolution of the problem with a higher (smallest) value for ϵ becomes more time consuming (as observed for PSO-BA). On the other hand, the majority of the configurations published by Birgin and Sobral are wrong for the accuracy $\epsilon = 10^{-9}$; in this case, the goal of the following experimental part (on Set2) in given in order to show how PSO-BA is able to provide 50% of new bounds for the instances of Set2 even the precision ϵ is more tight. From Table 7.6, one can observe what follows:

				BS		PSO-BA		
Inst.	n	H	W	L_{BS}	t_{BS}	$L_{\text{PSO-BA}}$	$t_{\text{PASO-BA}}$	%Imp
Inst_23	23	9.5	9.5	2.51595165	189.50	2.5216323421	7121.41	-0.22579
Inst_24	24	9.5	9.5	2.62319770	4408.99	2.6236233008	5221.52	-0.01622
Inst_25	25	9.5	9.5	2.69564484	0.90	2.6959603865	6705.21	-0.01171
Inst_26	26	9.5	9.5	2.96170666	297.13	2.9630848823	4958.75	-0.04653
Inst_27	27	9.5	9.5	3.06844011	191.42	3.0712826787	5171.13	-0.09264
Inst_28	28	9.5	9.5	3.11962555	2711.62	3.1356918085	5233.56	-0.51501
Inst_29	29	9.5	9.5	3.19136325	1407.72	3.1842350645	7087.69	**0.22336**
Inst_30	30	9.5	9.5	3.20868939	101.87	3.2088350642	6998.76	-0.00454
Inst_31	31	9.5	9.5	3.28596422	319.28	3.2856970017	6046.68	**0.00813**
Inst_32	32	9.5	9.5	3.30869493	5308.01	3.3081236222	1644.42	**0.01727**
Inst_33	33	9.5	9.5	3.31783818	333.14	3.3161653650	2435.56	**0.05042**
Inst_34	34	9.5	9.5	3.320476891	2877.74	3.3182590212	1952.50	**0.06679**
Inst_35	35	9.5	9.5	3.322499194	189.94	3.3217987089	7133.50	**0.02108**
Inst_36	36	9.5	9.5	3.322770105	155.49	3.3228712900	37762.00	-0.00305
Inst_37	37	9.5	9.5	3.503079169	2410.46	3.4958062319	6882.81	**0.20762**
Inst_38	38	9.5	9.5	3.547458805	2032.84	3.5462125173	7197.73	**0.03513**
Inst_39	39	9.5	9.5	3.603308794	3819.94	3.5895673155	7034.14	**0.38136**
Inst_40	40	9.5	9.5	3.642753852	3179.57	3.5955038566	5026.46	**1.29710**
Inst_41	41	9.5	9.5	3.722102153	1029.65	3.7161287915	3270.89	**0.16048**
Inst_42	42	9.5	9.5	3.853666642	582.70	3.8376097707	4667.00	**0.41666**
Inst_43	43	9.5	9.5	3.924985274	1520.75	3.9248156942	4416.17	**0.00432**
Inst_44	44	9.5	9.5	3.986787068	748.94	3.9886070286	6434.89	-0.04565
Inst_45	45	9.5	9.5	4.130763956	746.93	4.1306583614	5320.36	**0.00256**
Inst_46	46	9.5	9.5	4.265016322	4779.91	4.2665891003	6151.17	-0.03688
Inst_47	47	9.5	9.5	4.391983174	74.73	4.3943264586	6614.22	-0.05335
Inst_48	48	9.5	9.5	4.470228233	675.51	4.4905968587	7157.12	-0.45565
Inst_49	49	9.5	9.5	4.547960695	4383.69	4.5860314095	5439.59	-0.83709
Inst_50	50	9.5	9.5	4.636149740	363.37	4.6172534818	6180.00	**0.40759**
Inst_55	55	9.5	9.5	4.841544131	12954.13	4.8105482954	6779.32	**0.64021**
Inst_60	60	9.5	9.5	5.311832058	1099.51	5.1910141534	7038.33	**2.27451**
Inst_65	65	9.5	9.5	5.727987481	324.60	5.7606785423	6026.08	-0.57073
Inst_70	70	9.5	9.5	5.968047356	93.59	5.9686269702	5398.85	-0.00971
Inst_75	75	9.5	9.5	6.403245161	859.05	6.3890257069	6944.62	**0.22207**
Inst_80	80	9.5	9.5	6.961754027	8689.86	6.8890036929	2822.65	**1.04500**
Inst_85	85	9.5	9.5	7.290805951	11636.12	7.3997364732	7061.91	-1.49408
Inst_90	90	9.5	9.5	7.468834922	1039.84	7.7968387460	6261.01	-4.39163
Inst_95	95	9.5	9.5	7.999432983	1301.22	7.9902058321	6849.88	**0.11535**
Inst_100	100	9.5	9.5	8.504432593	3346.60	8.5998068889	6805.00	-1.12147
Av.%				4.3675532426	2268.06	4.3750645451	6559.29	-0.06144

Table 7.6: Behavior of PSO-BA versus BS on instances of Set2

1. First, PSO-BA is able to achieve 50% of instances of Set2 (as represented in the bold, the last column). These bounds are provided with a more tight accuracy, i.e. 10^{-9} almost of 10^{-4}.

2. Second, among the rest of the unmatched bounds, the values obtained by PSO-BA remain very close to that obtained by BS, although most of the solutions remain in feasible for $\epsilon = 10^{-9}$.

3. Third and last, when comparing PSO-BA's average runtime to that BS, one can observe that BS is three times faster that PSO-BA. Of course, it can be explained by the fact that the accuracy used by PSO-BA is more tight and interesting and

	Particle Swarm Optimization-Based Algorithm: PSO-BA					
Inst.	Min	t	Mean	t	Max	t
Inst_23	2.5216323421	7121.41	2.5368880003	5355.428	2.5639770917	3723.66
Inst_24	2.6236233008	5221.52	2.6398844469	4553.006	2.6537342444	1095.88
Inst_25	2.6959603865	6705.21	2.7222414461	11428.754	2.7839797358	3234.53
Inst_26	2.9630848823	4958.75	2.9837655111	6319.012	3.0043271543	6456.09
Inst_27	3.0712826787	5171.13	3.0831767745	6142.517	3.0956795182	6953.37
Inst_28	3.1356918085	5233.56	3.1439374706	6952.967	3.1580436895	6723.78
Inst_29	3.1842350645	7087.69	3.1934692645	6697.26	3.2216232288	7061.74
Inst_30	3.2088350642	6998.76	3.2495510792	5454.3321	3.2817067329	7127.8
Inst_31	3.2856970017	6046.68	3.307203712	11182.261	3.363613535	7168.8
Inst_32	3.3081236222	1644.42	3.313378479	10133.048	3.3182645399	6137.63
Inst_33	3.3161653650	2435.56	3.3184625005	6264.42	3.3217032273	7150.68
Inst_34	3.3182590212	1952.50	3.3215400041	6272.345	3.3228756642	6820.92
Inst_35	3.3217987089	7133.50	3.3332331142	5893.023	3.4327970651	5630.98
Inst_36	3.3228712900	37762.00	3.3230344505	13783.321	3.3243915348	5644.7
Inst_37	3.4958062319	6882.81	3.5400127576	6046.534	3.6017111723	6710.91
Inst_38	3.5462125173	7197.73	3.5820165362	6168	3.6052739647	5275.36
Inst_39	3.5895673155	7034.14	3.5909626905	5806.733	3.5928850143	3322.78
Inst_40	3.5955038566	5026.46	3.6452285223	5491.837	3.7237035617	6494.14
Inst_41	3.7161287915	3270.89	3.7564769655	6001.118	3.8519263896	5554.35
Inst_42	3.8376097707	4667.00	3.8693661043	5618.583	3.9152955172	5759.36
Inst_43	3.9248156942	4416.17	3.9749799671	5831.878	3.9991143903	6918.36
Inst_44	3.9886070286	6434.89	4.0131704006	5841.446	4.1180969516	5163.55
Inst_45	4.1306583614	5320.36	4.1782064696	6212.231	4.2849995189	6981.31
Inst_46	4.2665891003	6151.17	4.3283727022	6354.53	4.3678508588	5574.48
Inst_47	4.3943264586	6614.22	4.4623625614	7419.6	4.5701405201	6877.02
Inst_48	4.4905968587	7157.12	4.5198937529	6278.572	4.5836754426	7054.4
Inst_49	4.5860314095	5439.59	4.619693128	6349.459	4.6626860167	6588.9
Inst_50	4.6172534818	6180.00	4.6418639585	5967.763	4.7277904183	6156.11
Inst_55	4.8105482954	6779.32	4.8299036624	6628.282	4.9867852143	7027.38
Inst_60	5.1910141534	7038.33	5.2512984254	6242.131	5.3828178819	4925.11
Inst_65	5.7606785423	6026.08	5.8385142213	6039.088	5.9474071446	4466.7
Inst_70	5.9686269702	5398.85	6.2450611819	5699.728	6.3680375088	5593.33
Inst_75	6.3890257069	6944.62	6.4973116491	6486.021	6.7400939195	5247.02
Inst_80	6.8890036929	2822.65	7.009029313	5279.314	7.0937291177	6550.42
Inst_85	7.3997364732	7061.91	7.4607755765	12175.776	7.5652610868	6820.82
Inst_90	7.7968387460	6261.01	7.9602731992	5712.088	8.0304486591	5617.68
Inst_95	7.9902058321	6849.88	8.2192708311	6202.911	8.4228278286	5611.02
Inst_100	8.5998068889	6805.00	8.6793850257	5453.895	8.8052406554	4787.86
Av %	4.3750645451	6559.29	4.4258735752	6782.61	4.4945925188	5842.34

Table 7.7: Variation of the quality of the bounds achieved by PSO-BA (with their runtimes) for the ten trials on instances of Set2.

therefore, inducing a less convergence of the algorithm.

Because PSO-BA considered ten trials for each instance of Set2, Table 7.5 reports the worst, best and average upper bounds achieved by PSO-BA for these ten trials and their corresponding runtimes (fixed to a maximum value of two hours).

7.3.2 The second version of the problem: S-ISP container case

In this section, the performance of PSO-BA is evaluated on a set containing 48 instances extracted from M'Hallah *et al.* [55], where the number of spheres to pack varies from 3 to 50.

The rest of this part is composed of two sections. The first section discusses the parameter settings used by PSO-BA. Second and last, a comparative study is made between the results achieved by PSO-BA and those realized by M'Hallah *et al.*'s method (noted VNS).

Parametre settings

As for the first version of the problem (i.e., O-SIP, Section 7.3.1), PSO-BA uses some random strategies for building the starting population and its updating. For each run, several solutions can be provided and so, ten trials are also considered for the O-SIP. We recall that PSO-BA involves several parameters, like c_1, c_2 and the size K of the population. Several adjustments have been considered for both c_1 and c_2. Limited computational results showed that $c_1 = c_2 = 2.05$ remains also the best tuning for the instances tested; it confirms the standard values used in the literature for such version of PSO.

With the aforementioned values, PSO-BA is called with a population with different sizes, where K varies in the discrete intervalle $\{10; 20; 30; 40; 50; , 60; 70; 80\}$. Such a variation has been used for three instances tested in M'Hallah *et al.* [55], where the number of spheres varies from 3 (small-sized instance) to 50 (large-sized instance). For each run, the runtime limit was fixed to 7200 seconds.

	Variation of the size K of the population			
#Inst.	10	20	30	40
3	2.15506490414077	2.15476096191240	2.15470186648428	2.15470054745249
20	3.74561767422006	3.47491673177999	3.47393601111318	3.47482020706182
50	4.62028414558653	4.56457920927597	4.55705507103570	4.55179457307563
%Average	3.50698890798245	3.39808563432279	3.39523098287772	**3.39377177586331**
	Variation of the size K of the population			
#Inst.	50	60	70	80
3	2.15470054655810	2.15470053988408	2.15470053839904	2.15470053839954
20	3.47380714430956	3.47441185776681	3.47362514648301	3.47388369912280
50	4.56043890704175	4.55966448114631	4.56595495016317	4.56534705871463
%Average	3.39631553263647	3.39625895959906	3.39809354501507	3.39797709874566

Table 7.8: Behavior of PSO-BA when varying the size K of the population

For these instances, Table 7.8 shows the average results (bounds) achieved by PSO-BA

over the ten trials (column 1 reports the instance information where the value corresponds to the instance containing 3, 20 and 50 spheres to pack and, the other columns display the bounds realized by PSO-BA fo each value associated to the size of the population).

By analyzing the results of Table 7.8, one can observe that the best average results are realized globally for $K = 40$. Hence, for the rest of the experimental part, c_1 and c_2 are fixed to 2.05 (other values of c_1 and c_2 provided almost the same bounds), the size K of the population is fixed to 40 and the runtime limit is fixed to 7200 second.

All results achieved by both VNS and PSO-BA are reported in Table 7.9. Column 1 shows the number of spheres representing each instance, column 2 displays VNS's sphere (noted S_{VNS}) (taken from M'Hallah *et al.* [55]), column 3 tallies the upper bound $S_{\text{PSO}-BA}$ achieved by PSO-BA whereas column 4 reports the percentage improvement %Imp (when it happens) yielded by PSO-BA when compared to the results reached by VNS (%Imp $= 100 \times \frac{S_{\text{VNS}} - S_{\text{PSO-BA}}}{S_{\text{VNS}}}$).

The analysis of the results of Table 7.9 follows:

1. First, on the one hand, PSO-BA is able to improve 15 new upper bounds out of 48. In this case, it realizes a percentage of 31.25% of new bounds for all tested instances. On the other hand, it matches 19 instances out of 48, which represents a percentage of 39.58% of the instances tested. However, it fails to match the rest of the instances (representing 14 instances out of 48), PSO-BA's bounds remain very close to that of VNS.

2. Second, when comparing PSO-BA's results to VNS's ones, one can observe that the percentage of the improvement (among the improved bounds) varies from 5.0182081E-14 (instance containing 13 spheres) to 7.0168322E-03 (instance containing 43 spheres). It means that the performance of PSO-BA doesn't depend on the instance size, but rather it has a good behavior on any type of instances.

3. Third and last, PSO-BA outperforms VNS since it is able to achieve a better global average bound; that is equal to 3.62719494740341 while VNS achieves only a global average bound of 4.55221146893267. Such an average value realizes an average percentage improvement of 3.63744096493292, as declared in the last column an line of Table 7.9.

We have already mentioned that ten trials of PSO-BA are considered for each instance of 48 instances tested in this part. Table 7.10 reports the worst, average and best upper bounds achieved by PSO-BA and their corresponding runtimes (fixed to a maximum value of two hours).

Inst	VNS	PSO-BA	gap
3	2.15470053824456	2.15470053837925	-1.3469004E-10
4	2.22474487138667	2.22474487139159	-4.9200644E-12
5	2.41421356236813	2.41421356237309	-4.9600324E-12
6	2.41421356236983	2.41421356237309	-3.2600589E-12
7	2.59125387233696	**2.59125387233612**	**8.4021679E-13**
8	2.64532877601513	2.64532877601607	-9.4013686E-13
9	2.73205080756539	**2.73205080756522**	**1.6964208E-13**
10	2.83246456105391	2.83246456105391	0.0000000E+00
11	2.90211303259014	2.90211303259014	0.0000000E+00
12	2.90211303258981	2.90211303258981	0.0000000E+00
13	3.00000000000000	3.00000000000000	0.0000000E+00
14	3.09114544488862	**3.09114544488857**	**5.0182081E-14**
15	3.14164262494867	**3.14164262494856**	**1.1013412E-13**
16	3.21568303201004	3.21568303201004	0.0000000E+00
17	3.27124551170111	3.27124551170111	0.0000000E+00
18	3.31898878173418	3.31898878173418	0.0000000E+00
19	3.38601597327490	3.38601597327490	0.0000000E+00
20	3.47353896224525	3.47353896224525	0.0000000E+00
21	3.48635141041092	3.48635141041092	0.0000000E+00
22	3.57983319115373	3.57983319115373	0.0000000E+00
23	3.62751643653543	3.62751643653542	0.0000000E+00
24	3.68539493545674	**3.68539493545579**	**9.5035091E-13**
25	3.68742674748920	3.68742674748920	0.0000000E+00
26	3.74740577652751	3.74740577652751	0.0000000E+00
27	3.81341595688139	3.81341595688139	0.0000000E+00
28	3.84164027814771	3.84164027814771	0.0000000E+00
29	3.87708910315835	**3.87708910315565**	**2.7000624E-12**
30	3.91649166155428	3.91649168869767	-2.7143390E-08
31	3.95075448490565	3.95075448492430	-1.8650415E-11
32	3.98744038931492	3.98744038931492	0.0000000E+00
33	4.01990091595853	4.01990092544962	-9.4910897E-09
34	4.04771997123059	4.04771997123695	-6.3602457E-12
35	4.08440574075338	4.08440776576801	-2.0250146E-06
36	4.11298932968653	**4.11298932968421**	**2.3199220E-12**
37	4.15478125199121	4.15480304173093	-2.1789740E-05
38	4.15766926004822	4.15767242127718	-3.1612290E-06
39	4.22394975626618	**4.22394975626377**	**2.4096281E-12**
40	4.25533295365295	4.25533314299430	-1.8934135E-07
41	4.25533295365297	4.25533295365297	0.0000000E+00
42	4.25533295365297	4.25533295365297	0.0000000E+00
43	4.36004332088190	**4.35302648869109**	**7.0168322E-03**
44	4.38283083788345	4.38284540902735	-1.4571144E-05
45	4.41007453378598	**4.40700316053521**	**3.0713733E-03**
46	4.44190731948126	**4.44178542926357**	**1.2189022E-04**
47	4.47455137992898	**4.47446547698958**	**8.5902939E-05**
48	4.49712009262402	**4.49632831111380**	**7.9178151E-04**
49	4.52594554850145	**4.51928918657509**	**6.6563619E-03**
50	4.55221146893267	**4.55095440529171**	**1.2570636E-03**
%Av.	3.62758993557859	**3.62719494740341**	**3.94988175E-04**

Table 7.9: Behavior of PSO-BA versus VNS

Inst.	Min	T_s	Mean	T_s	Max	T_s
3	2.15470053837925	343.4	2.15470054745249	696.18	2.15470058855811	759.41
4	2.22474487139159	93.12	2.22474487139159	148.41	2.22474487139159	200.09
5	2.41421356237309	473.91	2.41421356237313	689.92	2.41421356237328	638.23
6	2.41421356237309	66.88	2.41421356237309	84.38	2.41421356237309	100.77
7	2.59125387233612	827.55	2.59125387234287	872.48	2.59125387236960	917.7
8	2.64532877601607	310.62	2.64532877601610	390.74	2.64532877601627	489.64
9	2.73205080756522	206.78	2.73205080757861	400.98	2.73205080760768	482.46
10	2.83246456105391	1290.2	2.83246456105394	1,528.77	2.83246456105402	1220.28
11	2.90211303259014	153.82	2.90211303259042	182.37	2.90211303259129	208.09
12	2.90211303258981	1179.29	2.93147989119104	1,562.56	3.00000744211427	589.34
13	3.00000000000000	488.9	3.00000000000000	1,892.93	3.00000000000000	3592.2
14	3.09114544488857	458.23	3.09114544488868	619.36	3.09114544488887	689.32
15	3.14164262494856	6282.91	3.14164262494862	654.29	3.14164262494873	688.91
16	3.21568303201004	1124.03	3.21575622434419	3,803.29	3.21601470312793	4195.9
17	3.27124551170111	92.19	3.27124551170111	98.42	3.27124551170111	115.44
18	3.31898878173418	707.66	3.31898878173422	1,280.04	3.31898878173439	804.43
19	3.38601597327490	6738.12	3.38601613073694	6,843.07	3.38601739312740	6967.01
20	3.47353896224525	7000.69	3.47482020706182	7,136.55	3.48635141041092	7081.45
21	3.48635141041092	2732.54	3.48635141041092	14,531.87	3.48635141041092	6428.52
22	3.57983319115373	2500.96	3.57985115336840	5,448.22	3.57993688485445	6331.79
23	3.62751643653542	5236.08	3.62844254665527	4,802.72	3.63640188028221	5502.21
24	3.68539493545579	6538.98	3.68615979183021	5,955.63	3.69110138973029	7012.51
25	3.68742674748920	2661.64	3.68742674748920	4,725.19	3.68742674748920	7192.52
26	3.74740577652751	1805.77	3.74740793153699	5,401.95	3.74742732662231	5749.9
27	3.81341595688139	4003.44	3.81544731674577	6,093.95	3.81999410151570	7108.87
28	3.84164027814771	6877.24	3.84272704790780	6,901.47	3.84558183473812	7116.96
29	3.87708910315565	6802.59	3.87730118326704	6,935.02	3.87846957514673	7049.88
30	3.91649168869767	6661.05	3.91829081798393	6,880.67	3.93405466697792	7049.88
31	3.95075448492430	6794.01	3.95086041113647	6,175.04	3.95134072468805	5992.75
32	3.98744038931492	6783.05	3.98851394803272	7,004.96	3.99817438478019	7168.49
33	4.01990092544962	6897.21	4.02005383864234	6,671.97	4.02140153547888	6942.3
34	4.04771997123695	5846.87	4.04927576754548	6,305.91	4.05745310904699	5933.43
35	4.08440776576801	6852.7	4.08545923767508	6,928.40	4.09439687162367	7019.74
36	4.11298932968421	5000.74	4.12848947081492	6,403.72	4.22502558279313	6607.45
37	4.15480304173093	7045.52	4.15645995438667	6,629.09	4.15910845682143	5171.55
38	4.15767242127718	6661.83	4.15782908980364	6,610.93	4.15846947834974	6349.76
39	4.22394975626377	6833.07	4.22710896351551	5,967.81	4.23683285110548	5556.08
40	4.25533314299430	6588.66	4.25685927331679	6,555.34	4.26319122701637	6712.78
41	4.25533295365297	7198.97	4.28448631182666	6,475.96	4.29781749796656	6826.14
42	4.25533295365297	5984.61	4.28935477133131	6,521.89	4.30814247814195	6119.67
43	4.35302648869109	5153.7	4.35769097669484	6,213.76	4.36339912798064	5096.6
44	4.38284540902735	6339.39	4.38458145671599	6,097.95	4.38844088733059	5499.41
45	4.40700316053521	6762.92	4.41186736346516	6,834.68	4.42495522072477	7185.73
46	4.44178542926357	6146.29	4.44491292847574	6,601.75	4.45470625669417	6453.07
47	4.47446547698958	5523.97	4.47692358432714	6,441.80	4.48121969144357	5589.82
48	4.49632831111380	5194.01	4.49757252031241	6,031.53	4.50295464097537	6805.06
49	4.51928918657509	7102.21	4.52176959805031	6,593.10	4.52713584368006	6089.51
50	4.55095440529171	6346.64	4.55179457307563	6,602.43	4.55375768598201	6977.97
Av.%	3.62719494740341	4,181.56	3.63028017491915	4,713.11	3.63744096493292	

Table 7.10: Solutions' quality (with their runtimes) achieved by PSO-BA for the ten trials

7.4 Conclusion

In this chapter, the three-dimensional identical sphere packing problem is solved by using a particle swarm optimization-based algorithm. The proposed method is based upon a

greedy truncated tree search procedure which served to generate a starting population. At each step of the algorithm, the unfeasible solutions are repaired when its associated particle is positioned out of the feasible region. In this case, a continuous local optimization is applied. The proposed method is also based upon the version of particle swarm that is able to ensure its convergence to a final feasible solution. The behavior of the proposed method was evaluated on a set of benchmark instances available in the literature, where its obtained results were compared to those reached by two recent methods available in the literature. As reported in the experimental part, the proposed method remains competitive by providing new bounds.

Chapter 8

Conclusions and perspetives

8.1 Conclusion

Cutting and Packing (C&P) problems are encountered in numerous industrial domains such as transportation, logistics, reliability, and production. They appear either as standalone problems or as subproblems of more complex combinatorial and continuous mathematical programming models. Because of their wide range of applicability, these problems have known a large number of variations such as: (C&P) circles / rectangles into a single open / close container, (C&P) circles / rectangles into multiple containers, (C&P) irregular shapes into (multiple) container(s), (C&P) into single and multiple containers with specific technological constraints, (C&P) with disjunctive constraints, single and multiple objective (C&P), integer, linear, non-linear, deterministic and stochastic, etc. Despite the technological and knowledge advance, solving (C&P) problems remains a challenging problem. "Efficient" approaches-based resolution techniques have been and are being proposed for the different variants of this problem.

Our contributions are divided into four chapters.

The first contribution was described in chapter 4, where the three-dimensional sphere packing problem is solved by using a dichotomous search-based heuristic. The proposed method is based upon three complementary phases: (i) a greedy selection phase which tries to select many eligible positions to iteratively packing all predefined items into the target object, (ii) a width beam search phase that serves to explore some promising paths, and (iii) a dichotomous search that serves to diversify the search space. The first two phases iterated until reaching the final objet; that is, the container with the smallest length containing all the items. The performance of the proposed method is evaluated on benchmark instances of the literature where the provided results are compared to those reached by some recent methods of the literature. The proposed method remains competitive and succeeded in yielding new solutions on many instances.

Chapter 5 contains the second contribution: we investigated the use of the global dichotomous search-based heuristic for approximately solving the three-dimensional sphere packing problem The proposed method is based on the three features: (i) creating a subset of paths by applying an extended greedy procedure, (ii) using a local operator in order to estimate lower bounds associated to internal nodes and (iii) applying an iterative search for diversifying the search process. The performance of the proposed algorithm

was evaluated on benchmark instances taken from the literature and the results reached by the method were compared to those reached by recent methods available in the literature. The proposed method succeeded to all methods in yielding new solutions for most of instances.

In chapter 6, the third contribution is presented, where the three-dimensional sphere packing is tackled. The problem consists in finding the greatest density of a (sub)set of predefined spheres (objects) into a cube (parallelepiped). We solved it by applying a cooperative method that combines three main features: (i) a best-local position procedure stage, (ii) an intensification stage and (iii) a diversification stage. The first stage ensures a starting feasible solution using a basic greedy local strategy. The second stage tries to solve a series of decision problems in order to place a subset of complementary spheres. The third stage forces the current solution to remove some packed items and then replaced them with other promising spheres. The performance of the proposed method is evaluated on benchmark instances taken from the literature where the achieved results are compared to those reached by some recent methods of the literature. The proposed method remains competitive and succeeded in yielding new solutions for many instances.

The last contribution of the thesis, on the three-dimensional identical sphere packing problem, is described in chapter 7. We solved such a problem by using a particle swarm optimization-based algorithm, which is based upon a greedy truncated tree search procedure that serves to generate a starting population. At each step of the algorithm, the unfeasible solutions are repaired when its associated particle is positioned out of the feasible region. In this case, a continuous local optimization was applied. The proposed method is also based upon the version of particle swarm that is able to ensure its convergence to a final feasible solution. The behavior of the proposed method was evaluated on a set of benchmark instances available in the literature, where its obtained results have been compared to those reached by two best methods of the literature. As reported in the experimental study, the proposed method remains competitive by providing new bounds.

8.2 Perspectives

The work presented in this thesis could be adapted for tackling variants of packing problems. Indeed, several of the proposed methods can be adapted for solving the following problems:

- the two-dimensional circular and rectangular packing problems, where the container may be a rectangle or a strip,
- the irregular packing problem into a container,
- both two and three dimensional packing problems with multiple containers and,

- the capacitated vehicle routing problem with two and three dimensional loading constraints.

According to the average running time needed by almost of the used methods, we believe that parallel versions of these methods can be envisaged. Indeed, we believe that type of methods could be useful for reducing the runtime and increasing the quality of the solutions. In fact, a parallel cooperative particle swarm optimization seems a good choice, where each processor may be assigned to a predefined swarm and all processors simultaneously synchronize the solutions reached up to now.

Related publications

A- International journals:

1- Mhand Hifi and Labib Yousef (2017). A global dichotomous search-based heuristic for the three-dimensional sphere packing problem. International Journal of Operations Research, to appear.

2- Mhand Hifi and Labib Yousef (2017). A Hybrid Algorithm for Packing Identical Spheres into a Container. Expert Systems With Applications (submitted)

3- Mhand Hifi and Labib Yousef (2017). A Cooperative Method for the Sphere Packing Problem. European Journal of Operational Research (under revision)

4- Mhand Hifi and Labib Yousef (2015). A dichotomous search-based heuristic for the three-dimensional sphere packing problem. Cogent Engineering, 2(1): Article 994257 (http://dx.doi.org/10.1080/23311916.2014.994257).

B- International Conferences :

1- Mhand Hifi, Dominique Lazure and Labib Yousef (2017). Solving the Identical Sphere Packing Problem with the Particle Swarm Optimization-Based Approach. Proceedings of the 47th International Conference on Computers & Industrial Engineering (CIE47), October 11-13 (accepted).

2- Mhand Hifi, Dominique Lazure and Labib Yousef (2017). Solving Packing Identical Spheres into a Smallest Sphere with a Particle Swarm Optimization. Proceedings of the 4rd International Conference on Control, Decision and Information Technologies (CoDIT17), April 5-7. (Forthcoming).

3- Mhand Hifi and Labib Yousef (2014). Width beam and hill-climbing strategies for the three-dimensional sphere packing problem. In Proc. of IEEE, Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 421–428.

C- Book Chapter:

1- Mhand Hifi and Labib Yousef (2016). Handling lower bound and hill-climbing strategies for sphere packing problems. Chapter in Recent Advances in Computational Optimization, pp. 145–164. Springer.

D- National Conferences :

1- Mhand Hifi and Labib Yousef (2017). Optimisation par essaims particulaires pour un problème de placement de sphéres. ROADEF 2017, 22-24 février, France.

2- Mhand Hifi and Labib Yousef (2016). Une Heuristique pour le Placement de Sphéres dans un Container.ROADEF,10-12 février, Compiègne, France. Metz. France.

3- Mhand Hifi and Labib Yousef (2015). Une Méthode Heuristique pour le Problème de Placement de Sphéres dans un Container.ROADEF, 25-27 février, Marseille, France.

Bibliography

[1] A. Abdelhakim and M. Boudhar. Parallel branch-and-bound and parallel pso algorithms for job shop scheduling problem with blocking. *International Journal of Operational Research*, 16(1):14–37, 2013. (Cited in page 76.)

[2] H. Akeb. A look-ahead-based heuristic for packing spheres into a bin: The knapsack case. *Procedia Computer Science*, 65:652–661, 2015. (Cited in pages 42, 107, 108 and 110.)

[3] H. Akeb. A two-stage look-ahead heuristic for packing spheres into a three-dimensional bin of minimum length. In *Recent Advances in Computational Optimization*, volume 610, pages 127–144. Springer International Publishing, 2016. (Cited in page 42.)

[4] H. Akeb and M. Hifi. Algorithms for the circular two-dimensional open dimension problem. *International Transactions in Operational Research*, 15:685–704, 2008. (Cited in pages 42 and 61.)

[5] H. Akeb and M. Hifi. An adaptive look-ahead strategy-based algorithm for the circular open dimension problem. In *The Second International Conference on Adaptive and Self-Adaptive Systems and Applications*, volume 7, pages 158–163, 2010. (Cited in page 42.)

[6] H. Akeb, M. Hifi, and R. M'Hallah. A beam search algorithm for the circular packing problem. *Computers & Operations Research*, 36(5):1513–1528, 2009. (Cited in page 42.)

[7] H. Akeb, M. Hifi, and S. Negre. An augmented beam search-based algorithm for the circular open dimension problem. *Computers & Industrial Engineering*, 61(2):373–381, 2011. (Cited in page 42.)

[8] E.G. Birgin and F.N.C Sobral. Minimizing the object dimensions in circle and sphere packing problems. *Computers & Operations Research*, 35(7):2357–2375, 2008. (Cited in pages 38, 39, 40, 68, 69, 70, 85, 86, 88, 133 and 135.)

[9] C. Blum, J. Puchinger, G.R Raidl, and A Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011. (Cited in page 43.)

[10] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. (Cited in page 43.)

[11] I Boussaïd, J Lepagnot, and P Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82 – 117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations. (Cited in pages 43, 44, 49 and 50.)

[12] G.J. Cheng, A.B. Yu, and P. Zulli. Evaluation of effective thermal conductivity from the structure of a packed bed. *Chemical Engineering Science*, 54(19):4199 – 4209, 1999. (Cited in pages 11 and 26.)

[13] V. Claudio, S. Ismael, C. Rolando, and S Leszek. Secure wireless communication channel by the combination of sphere packing, channel code and space time. *WSEAS Transactions on Mathematics*, 6(2):395–399, 2007. (Cited in pages 2, 12 and 27.)

[14] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. (Cited in page 125.)

[15] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1):58–73, 2002. (Cited in page 125.)

[16] Y. Collette and P. Siarry. *Optimisation multiobjectif*. Editions Eyrolles, 2002. (Cited in pages 6 and 22.)

[17] J.H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013. (Cited in pages 2, 9, 12, 24, 25 and 27.)

[18] F. de Larrard and M. Buil. Granularité et compacité dans les matériaux de génie civil. *Materials & Structures*, 20(2):117–126, 1987. (Cited in pages 11 and 26.)

[19] F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10(1):89–104, 2004. (Cited in page 64.)

[20] J. Derbyshire. *Prime Obsession: Bernhard Riemann and the Greatest Unsolved Problem in Mathematics*. Plume, 2004. (Cited in pages 9 and 25.)

[21] H. Dyckhoff and U. Finke. *Cutting and packing in production and distribution: A typology and bibliography*. Springer Science & Business Media, 1992. (Cited in pages 7 and 23.)

[22] Z. Fu, W. Huang, and Z. Lü. Iterated tabu search for the circular open dimension problem. *European Journal of Operational Research*, 225(2):236–243, 2013. (Cited in page 46.)

[23] J. A. George, J. M. George, and B. W. Lamar. Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, 84(3):693–712, 1995. (Cited in pages 40 and 96.)

[24] F. Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990. (Cited in page 46.)

[25] R.L. Graham, B. D. Lubachevsky, and P. R. Nurmela, K. J .and Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181(1):139 – 154, 1998. (Cited in page 53.)

[26] D. Harald. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145 – 159, 1990. (Cited in pages 7 and 23.)

[27] M. Hifi, , and R. M'Hallah. Approximate algorithms for constrained circular cutting problems. *Computers & Operations Research*, 31(5):675 – 694, 2004. (Cited in page 42.)

[28] M. Hifi. An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization*, 46(8):1109–1122, 2014. (Cited in page 103.)

[29] M. Hifi and R. M'Hallah. Approximate algorithms for constrained circular cutting problems. *Computers and Operations Research*, 31(5):675–694, 2004. (Cited in page 50.)

[30] M. Hifi and R. M'Hallah. Beam search and non-linear programming tools for the circular packing problem. *International Journal of Mathematics in Operational Research*, 1(4):476, 2009. (Cited in pages 64 and 96.)

[31] M. Hifi and R. M'Hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009:1–22, 2009. (Cited in pages 12, 19, 27, 35, 38 and 40.)

[32] M. Hifi, R. M'Hallah, and T. Saadi. Algorithms for the constrained two-staged two-dimensional cutting problem. *INFORMS Journal on Computing*, 20(2):212–221, 2008. (Cited in pages 61 and 64.)

[33] M. Hifi and M. Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006. (Cited in page 103.)

[34] M. Hifi and T. Saadi. A cooperative algorithm for constrained two-staged two-dimensional cutting problems. *International Journal of Operational Research*, 9(1):104–124, January 2010. (Cited in pages 64 and 83.)

[35] M. Hifi, V. Th Paschos, and V. Zissimopoulos. A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*, 159(2):430 – 448, 2004. Supply Chain Management: Theory and Applications. (Cited in page 45.)

[36] M. Hifi and L. Yousef. Width beam and hill-climbing strategies for the three-dimensional sphere packing problem. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 421–428. IEEE, 2014. (Cited in pages 78, 86, 88, 98, 126 and 129.)

[37] M. Hifi and L. Yousef. A dichotomous search-based heuristic for the three-dimensional sphere packing problem. *Cogent Engineering*, 2(1):994257, 2015. (Cited in pages 78, 86, 88 and 98.)

[38] M. Hifi and L. Yousef. Handling lower bound and hill-climbing strategies for sphere packing problems. In *Recent Advances in Computational Optimization*, pages 145–164. Springer, 2016. (Cited in pages 78, 86, 88, 89, 92, 126, 127 and 129.)

[39] W. Q Huang, Y. Li, H. Akeb, and C. Lib. Greedy algorithms for packing unequal circles into a rectangular container. *Journal of the Operational Research Society*, 56(5):539–548, 2005. (Cited in pages 41 and 42.)

[40] W.Q Huang, Z.Z. Zeng, R.C. Xu, and Z.H Fu. Using iterated local search for efficiently packing unequal disks in a larger circle. In *Advanced Materials Research*, volume 430, pages 1477–1481. Trans Tech Publ, 2012. (Cited in page 49.)

[41] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. (Cited in page 52.)

[42] J Kennedy and R.C. Eberhart. Particle swarm optimization. page 1942–1948. Proceedings of IEEE International Conference on Neural Networks, 1995. (Cited in pages 50 and 124.)

[43] J. Kennedy and R. Mendes. Population structure and particle swarm performance. 2002. (Cited in page 52.)

[44] G. Kirkpatrick and P. V. Mario. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. (Cited in page 44.)

[45] T. Kubach, A. Bortfeldt, T. Tilli, and H. Gehring. *Parallel greedy algorithms for packing unequal spheres into a cuboidal strip or a cuboid*. Fernuniv., Fachbereich Wirtschaftswiss., 2009. (Cited in pages 12, 18, 28, 33, 42, 69, 70, 73, 86, 88, 89, 91, 92, 107, 112 and 113.)

[46] T. Kubach, A. Bortfeldt, T. Tilli, and H. Gehring. greedy algorithms for packing unequal spheres into a cuboidal strip or a cuboid. *Asia Pacific Journal of Operational Research*, 28(06):739 –753, 2011. (Cited in pages 12, 18, 19, 28, 33, 34, 42, 68, 69, 70, 73, 85, 86, 88, 107, 108 and 110.)

[47] S. Kucherenko, P. Belotti, L. Liberti, and N. Maculan. New formulations for the kissing number problem. *Discrete Applied Mathematics*, 155(14):1837 – 1841, 2007. (Cited in pages 9 and 25.)

[48] J.C. Lagarias. *The Kepler conjecture : the Hales-Ferguson proof*. Springer Science+Business Media, LLC, New York, NY, 2011. (Cited in pages 9 and 25.)

[49] J Leech. The problem of the thirteen spheres. *The Mathematical Gazette*, 40(331):22–23, 1956. (Cited in pages 9 and 25.)

[50] S.P. Li and K.k. Ng. Monte carlo study of the sphere packing problem. *Physica A: Statistical Mechanics and its Applications*, 321(1–2):359 – 363, 2003. Statphys-Taiwan-2002: Lattice Models and Complex Systems. (Cited in pages 10, 25 and 26.)

[51] Y. Li and W. Ji. Stability and convergence analysis of a dynamics-based collective method for random sphere packing. *Journal of Computational Physics*, 250:373–387, 2013. (Cited in page 122.)

[52] C.O. Lopez Soto. *Formulation space search for two-dimensional packing problems*. PhD thesis, Brunel University School of Information Systems, Computing and Mathematics Theses PhD Theses, 2013. (Cited in page 40.)

[53] B. D Lubachevsky and R.L. Graham. Curved hexagonal packings of equal disks in a circle. *Discrete & Computational Geometry*, 18(2):179–194, 1997. (Cited in page 53.)

[54] R. M'Hallah and A. Alkandari. Packing unit spheres into a cube using vns. *Electronic Notes in Discrete Mathematics*, 39:201 – 208, 2012. EURO Mini Conference. (Cited in pages 16, 31, 32 and 48.)

[55] R. M'Hallah, A. Alkandari, and N. Mladenovic. Packing unit spheres into the smallest sphere using vns and nlp. *Computers & Operations Research*, 40(2):603–615, 2013. (Cited in pages 48, 130, 132, 138 and 139.)

[56] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. (Cited in page 47.)

[57] N. Mladenović, F. Plastria, and D. Urošević. Formulation space search for circle packing problems. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 212–216. Springer, 2007. (Cited in page 40.)

[58] G. Moslehi, M. Mahnam, M. Amin-Nayeri, and A. Azaron. A branch-and-bound algorithm to minimise the sum of maximum earliness and tardiness in the single machine. *International Journal of Operational Research*, 8(4):458–482, 2010. (Cited in page 76.)

[59] V. Nguyen, J. Fortin, M. Guessasma, E. Bellenger, and C. Cogné. Heat transfer modeling by discrete element method. *Journal of the Mechanics of Materials and Structures*, 4:413–426, 2009. (Cited in pages 11 and 26.)

[60] V. Nguyen, J. Fortin, M. Guessasma, E. Bellenger, and C. Cogné. Thermomechanical modelling of friction effects in granular flows using the discrete element method. *Journal of Mechanics of Materials and Structures*, 4(2):413–426, 2009. (Cited in pages 11 and 26.)

[61] V. Nguyen, V. Magnier, J.F. Brunel, P. Dufrenoy, and P. Coorevits. Modelisation de la dissipation d energie d un freinage par la methode des élements discrets. In *10e colloque national en calcul des structures*, page Clé USB, Giens, France, 2011. (Cited in pages 11 and 26.)

[62] P. S. Ow and T. E. Morton. Filtered beam search in scheduling†. *The International Journal Of Production Research*, 26(1):35–62, 1988. (Cited in pages 42 and 64.)

[63] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In *In Proceedings of the Euro-International Symposium on Computational Intelligence 2002*, pages 214–220. IOS Press, 2002. (Cited in page 124.)

[64] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998. (Cited in pages 51 and 124.)

[65] M. Solimanpur, H. Sattari, and A. M. Abazari. Optimum process plan selection via branch-and-bound algorithm in an automated manufacturing environment. *International Journal of Operational Research*, 13(3):281–294, 2012. (Cited in page 76.)

[66] Y. Stoyan and G. Yaskov. *Packing Identical Spheres into a Rectangular Parallelepiped*, chapter Intelligent Decision Support: Current Challenges and Approaches, pages 47–67. Gabler, 2008. (Cited in pages 130, 132, 133 and 134.)

[67] Y. Stoyan, G. Yaskov, and G. Scheithauer. Packing of various radii solid spheres into a parallelepiped. *Central European Journal of Operations Research*, 11(04):389–407, 2003. (Cited in pages 19, 34, 68, 69, 70, 85, 86 and 88.)

[68] P. N Suganthan. Particle swarm optimiser with neighbourhood operator. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. (Cited in page 52.)

[69] A. Sutou and Y. Dai. Global optimization approach to unequal global optimization approach to unequal sphere packing problems in 3d. *Journal of Optimization Theory and Applications*, 114(3):671–694, 2002. (Cited in pages 2, 10, 25, 26, 60 and 122.)

[70] F. Taubert, M. W. Jahn, H. Dobner, T. Wiegand, and A. Huth. The structure of tropical forests and sphere packings. *Proceedings of the National Academy of Sciences*, 112(49):15125–15129, 2015. (Cited in pages 11 and 27.)

[71] S. Torquato. *Random heterogeneous materials: microstructure and macroscopic properties*, volume 16. Springer Science & Business Media, 2013. (Cited in page 1.)

[72] H. Wang, W. Huang, Q. Zhang, and D. Xu. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141(2):440–453, 2002. (Cited in pages 53, 54, 55, 97, 100 and 128.)

[73] J. Wang. Packing of unequal spheres and automated radiosurgical treatment planning. *Journal of Combinatorial Optimization*, 3(4):453–463, 1999. (Cited in pages 2, 10, 25, 26 and 60.)

[74] G. Wäscher, Haußner H., and H. Schumann. An improved typology of cutting and packing problems. *European journal of operational research*, 183(3):1109–1130, 2007. (Cited in pages 7, 8, 23, 24 and 60.)

[75] S. R. Williams and A. P. Philipse. Random packings of spheres and spherocylinders simulated by mechanical contraction. *Phys. Rev. E*, 67:051301, 2003. (Cited in pages 11 and 26.)

[76] M. Yavuz. Iterated beam search for the combined car sequencing and level scheduling problem. *International Journal of Production Research*, 51(12):3698–3718, 2013. (Cited in pages 64 and 83.)

[77] D.f. Zhang and A.S. Deng. An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Computers & Operations Research*, 32(8):1941–1951, 2005. (Cited in page 45.)

[78] Y. Zhizhong, Z.and Xinguo, H. Kun, H. Wenqi, and F. Zhanghua. Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*, 250(2):615 – 627, 2016. (Cited in page 47.)

[79] C. Zong. *Sphere packings*. Springer Science & Business Media, 2008. (Cited in pages 9 and 25.)