الجمهورية الديمقراطية الشعبية الجزائرية

République algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'enseignement supérieur et de la recherche scientifique

Nº Ref : ..... .

Centre Universitaire Abdelhafid Boussouf - Mila

Institut des Mathématiques et Informatique **Département Informatique**

Mémoire préparé en vue de l'obtention du diplôme de master en informatique

Spécialité : Intelligence artificielle et ses applications (I2A)

Une méthode de recherche tabou probabiliste pour le problème de localisation à deux niveaux avec contrainte de capacité

Présentée Par : Bellemalem Chahinez **&** Boudouda Taha Zackaria

Soutenue devant un jury composé de :

Encadrant :	Mr Oualid Guemri	Grade MCB
Président :	Mr Abdelkamel HETTAB	Grade MCB
Examinateur :	Mr Salim DJAABOUB	Grade MCB

Année universitaire : 2023/2024

Remerciement

Louange à ALLAH qui nous a donné la force et la volonté d'achever notre mémoire.

Nous tenons à exprimer notre profonde gratitude à notre professeur encadrant, Monsieur **Oualid GUEMRI**, dont la guidance éclairée et le soutien constant ont été essentiels tout au long de la création de ce mémoire. Sa passion pour le sujet et son engagement envers l'excellence académique ont été une source d'inspiration inestimable.

Nous adressons également nos sincères remerciements aux membres du jury, Monsieur **Abdelkamel HETTAB** et Monsieur **Salim DJAABOUB** pour leur disponibilité, leurs précieux conseils et leurs remarques constructives ont grandement enrichi notre travail.

Nous souhaitons aussi exprimer notre reconnaissance à l'ensemble du corps enseignant, leur dévouement à l'éducation et leurs efforts inlassables pour nous guider dans notre parcours académique ont joué un rôle déterminant dans notre réussite, leur maitrise et leur dévouement ont grandement contribué à notre compréhension approfondie des sujets abordés au cours de notre formation.

Nos remerciements s'étendent également à tous les membres du personnel administratif qui, par leur travail assidu en coulisses, ont créé un environnement propice à l'apprentissage et à la croissance intellectuelle.

Nous tenons également à exprimer notre reconnaissance envers nos parents, nos grands-parents, nos familles et nos amis, qui nous ont apporté un soutien moral précieux tout au long de ce parcours académique.

Enfin, nous voudrions adresser des remerciements spéciaux à ces personnes qui nous ont toujours soutenues, et ce malgré les circonstances. Nous tenons à exprimer notre gratitude à : **Khaled KHALFAOUI**, **Cheims-Eddine BOUDOUDOU, Skander BOUDOUDA, Sana KASSAMA, Randa BOUDOUDOU, Cheima Maram BOUDOUDA et Zineddine Abdellah Awab BOUDOUDA,** pour nous avoir toujours aidés dans les moments difficiles, envers et contre tous. Leurs conseils nous ont permis de faire un beau parcours, et de persévérer dans la vie.

Ce mémoire est le résultat d'un effort collectif, et nous sommes profondément reconnaissants envers chacune des personnes qui ont contribué à façonner cette expérience unique et enrichissante, merci à vous.

Table des matières

Table des matières II
Table des figures III
Liste des tableaux III
Liste des algorithmes IV
Acronymes V
Résumé VI
Introduction générale 1
Chapitre 1 : Définition et applications des problèmes de localisation 2
1.1 Introduction 2
1.2 Les problèmes de localisation 2
1.2.1 Problème de localisation sans contrainte de capacité 3
1.2.2 Problème de localisation avec contrainte de capacité 4
1.2.3 Problème du p-médiane 5
1.2.4 Problème de couverture maximale 5
1.2.5 Problème de localisation de hubs 5
1.2.6 Problèmes de localisation multi-produits 6
1.2.7 Problèmes de localisation multi-niveaux 6
1.2.8 Problèmes de localisation multi-objectif 7
1.3 Problème de localisation à deux niveaux avec contrainte de capacité 7
1.4 Applications des problèmes de localisation 8
1.5 Conclusion 9
Chapitre 2 : Méthodes et algorithmes d'optimisation 10
2.1 Introduction 10
2.2 Définition générale d'un problème d'optimisation 10
2.3 Méthodes approchées 10
2.3.1 Les Heuristiques 11
2.3.2 Les métaheuristiques 13
2.4 Méthodes Exactes 25
2.5 Méthodes hybrides 26
2.5.1 Combinaison des métaheuristiques avec des méthodes exactes 26
2.5.2 Combinaison des métaheuristiques avec d'autres métaheuristiques 27
2.6 Conclusion 28
Chapitre 3 : Une méthode de recherche tabou probabiliste pour résoudre le TSCFLP 29
3.1 Introduction 29
3.2 Définition du problème 29
3.3 Revue de la littérature 30
3.4 Algorithme proposé pour résoudre le TSCFLP 31
3.4.1 Représentation de la solution 32
3.4.2 Procédure de génération d'une solution initiale 32
3.4.3 Procédure de voisinage 33
3.4.4 Procédure d'allocation 36
3.4.5 Gestion de la liste tabou 37
3.5 Conclusion 38
Chapitre 4 : Tests et comparaison avec la littérature 39

4.1 Introduction 39
4.2 Description du jeu de données de la littérature 40
4.3 Résultats des tests 42
4.3.1 Paramètres et détails de l'implémentation 42
4.3.2 Structure de la solution 43
4.3.3 Résultats obtenus et comparaison avec la littérature 44
4.4 Conclusion 48
Conclusion générale 49
Bibliographie 50

Table des figures

Figure 1 : Représentation d'un problème de localisation générale 2
Figure 2 : Problème de localisation multi-niveaux 6
Figure 3 : Représentation d'un TSCFLP 7
Figure 4 : Le croisement à un point dans l'algorithme génétique 19
Figure 5 : Le croisement à deux points dans l'algorithme génétique 19
Figure 6 : Le croisement uniforme dans l'algorithme génétique 20
Figure 7 : La mutation dans l'algorithme génétique 20
Figure 8 : La sélection par roulette dans l'algorithme génétique 21
Figure 9 : Partie clients 41
Figure 10 : Partie entrepôts 41
Figure 11 : Partie usines 41
Figure 12 : Partie coût d'acheminement Entrepôts-Clients 41
Figure 13 : Partie coût d'acheminement Usines-Entrepôts 41
Figure 14 : Exemple de vérification manuelle d'une solution générée 44

Liste des tableaux

Tableau 1 : Paramètres utilisés pour la génération des instances 40
Tableau 2 : Structure d'une solution générée 43
Tableau 3 : Résultats obtenus pour le premier ensemble d'instances (50 usines, 100 entrepôts et 200 clients) 45
Tableau 4 : Résultats obtenus pour le deuxième ensemble d'instances (100 usines, 200 entrepôts et 400 clients) 45
Tableau 5 : Comparaison des meilleurs résultats obtenus avec ceux de la littérature 47

Liste des algorithmes

Algorithme 1 : Heuristique gloutonne de construction pour un probleme de minimisation.... 11
Algorithme 2 : Heuristique gloutonne randomisee de construction pour un probleme de minimisation 12
Algorithme 3 : Recherche locale 13
Algorithme 4 : Recuit simule 14
Algorithme 5 : Recherche tabou 15
Algorithme 6 : Recherche a voisinage variable 16
Algorithme 7 : GRASP 17
Algorithme 8 : L'algorithme génétique 18
Algorithme 9 : Optimisation par essaim de particules 23
Algorithme 10 : Optimisation par colonie de fourmis 23
Algorithme 11 : Colonie d'abeilles artificielles 24
Algorithme 12 : Optimisation par des baleines 25
Algorithme 13 : Branche and bound pour un probleme de minimisation 26
Algorithme 14 : Recherche tabou probabiliste pour le TSCFLP 32
Algorithme 15 : Generation d'une solution aleatoire realisable 33
Algorithme 16 : Procedure swap (solution,clients) 34
Algorithme 17 : Procedure add (solution,clients) 35
Algorithme 18 : Procedure drop (solution,clients) 35
Algorithme 19 : Procedure creerstructuredevoisinage (N(s) , L) 36
Algorithme 20 : Procedure d'allocation 37

Acronymes

GRASP : Greedy randomized adaptive search procedures.

GA : Genetic algorithm.

LS : Local search.

SA : Simulated annealing.

TS : Tabu search.

VNS : Variable neighborhood search.

PSO : Particle swarm optimization.

ACO : Ant colony optimization.

ABC : Artificial Bee Colony.

WOA : Whale Optimization Algorithm.

CS : Clustering search.

ALNS : Adaptive large neighborhood search.

LB : Local branching.

Résumé

Dans ce mémoire, nous présentons un algorithme de recherche tabou probabiliste pour résoudre le problème de localisation à deux niveaux avec contrainte de capacité (two-stage capacitated facility location problem, TSCFLP). Dans ce problème, un seul type de produit doit être transporté des usines aux clients, en passant par des entrepôts. Initialement, notre algorithme sélectionne aléatoirement des locaux (usines\ entrepôts) pour créer une solution initiale. Ensuite, il améliore itérativement cette solution en réduisant le coût total y compris le coût d'installation des locaux et le coût de transport. Enfin, les expérimentations montrent que l'algorithme proposé a donné des résultats prometteurs par rapport à la littérature.

Abstract

In this thesis, we present a probabilistic tabu search algorithm to solve the two-stage capacitated facility location problem (TSCFLP). In this problem, one kind of product needs to be transport from factories to customers through warehouses. Initially, our algorithm selects random facilities (factories/warehouses) to create an initial solution. Then, it improves iteratively this solution by reducing its total cost including location cost and transportation cost. Experiments show that the proposed algorithm reported promising results comparing to the literature.

ملخص

في هذه المذكرة، نقدم خوارزمية بحث طابو احتمالية لحل مشكلة تحديد المواقع ذات السعة المحدودة والموزعة على مستويين .ونقل نوع واحد من المنتجات من المصانع إلى الزبائن عبر المستودعات .تبدأ خوارزميتنا باختيار مواقع المصانع والمستودعات بشكل عشوائي لتشكيل حل ابتدائي .ثم تعمل هذه الخوارزمية على تحسين هذا الحل بطريقة تدريجية وذلك عن طريق تخفيض تكلفة اختيار المواقع وتكلفة النقل. أظهرت التجارب أن الخوارزمية المقترحة حققت نتائج واعدة بالمقارنة مع نتائج الخوارزمية المقترحة من قبل

Introduction générale

Les problèmes de localisation occupent une place importante dans la littérature de l'optimisation combinatoire en raison de leurs nombreuses applications. Le choix des locaux (usines, entrepôts, centres de distribution, etc.) est une question importante qui affecte directement la performance d'une entreprise, un mauvais choix peut donc entraîner des coûts élevés, une mauvaise satisfaction des clients et une utilisation inefficace des ressources. Par conséquent, il est nécessaire de développer et d'utiliser des algorithmes qui permettent de trouver des solutions bonne qualité pour ces problèmes en fonction des contraintes de chaque problème.

Parmi les problèmes de localisation les plus connues, on retrouve le problème de localisation à deux niveaux avec contrainte de capacité (TSCFLP). Dans celui-ci, les produits fabriqués par des usines sont transférés vers des entrepôts, ensuite les produits des entrepôts sont livrés aux clients. Le problème consiste à trouver la meilleure combinaison des locaux (entrepôts / usines) afin de satisfaire les clients et ce dans le but de réduire les frais d'ouverture des locaux, ainsi que les coûts de transport qui leurs sont associés.

Notre mémoire se concentre sur l'étude et la résolution du problème de localisation à deux niveaux avec contrainte de capacité. Et pour la première fois dans la littérature, nous proposons de résoudre ce problème en utilisant une nouvelle variante de la méthode de recherche tabou probabiliste (Probabilistic Tabu Search, PTS).

Nous organisons notre mémoire comme suit : dans le premier chapitre, nous allons présenter une vue d'ensemble des problèmes de localisation, leurs variantes et leurs applications. Puis, dans le deuxième chapitre, on va définir les problèmes d'optimisations, les méthodes et les algorithmes les plus connus pour les résoudre. Ensuite, dans le troisième chapitre, nous allons décrire en détail l'algorithme de recherche tabou probabiliste que nous avons proposé. Concernant le quatrième chapitre, on va évaluer l'efficacité de notre algorithme à travers des expériences et nous comparerons les résultats obtenus avec ceux de la littérature. Enfin, nous finirons par une conclusion.

Chapitre 1 : Définition et applications des problèmes de localisation

1.1 Introduction

Les problèmes de localisation sont parmi les problèmes les plus abordé dans la littérature de l'optimisation combinatoires. Ces problèmes possèdent de nombreuses applications dans la vie réelle. En plus, ils jouent un rôle important dans divers domaines tels que la logistique, la planification urbaine et la gestion des réseaux. Tout en contribuant à une distribution plus efficace des ressources, à une amélioration globale de la connectivité et de la prestation de services.

Dans ce premier chapitre, nous allons présenter les variantes des problèmes de localisation qui sont très proches du problème étudié dans ce mémoire [1, 2, 3].

1.2 Les problèmes de localisation

Dans les problèmes de localisation, on cherche à choisir un sous-ensemble de locaux (usines, entrepôts...etc.) parmi une liste de potentiel candidats. En plus, les locaux non sélectionnés sont alors rattachés aux locaux qui font partie du sous-ensemble sélectionné. En modifiant ce sous-ensemble via des opérations comme la permutation par exemple, on obtient une nouvelle solution de qualité différente. Prenons pour exemple le schéma suivant :

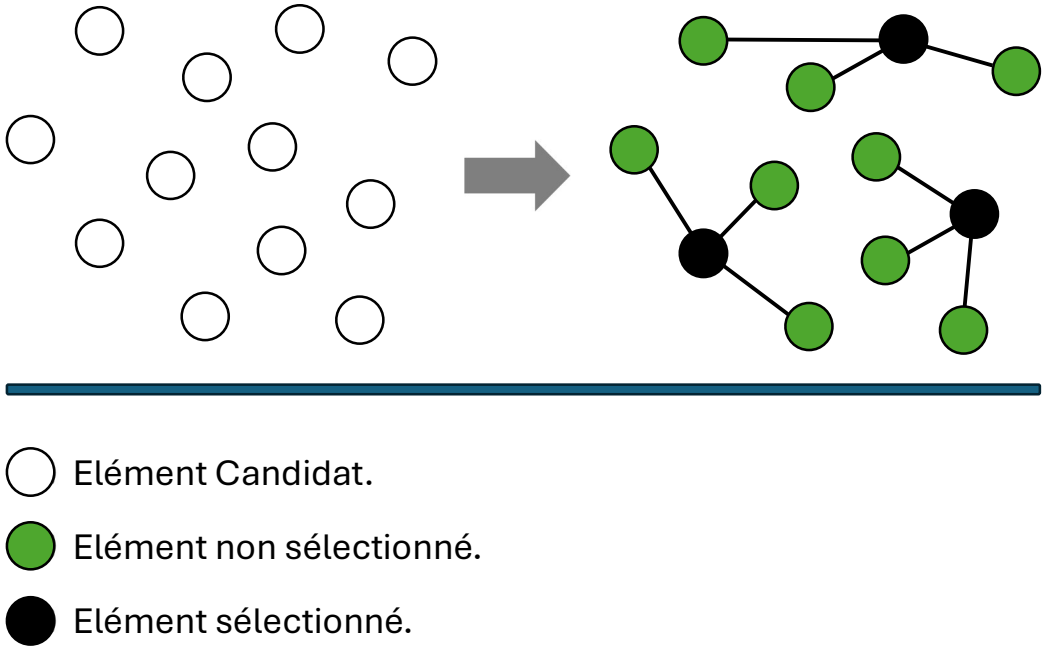


Figure 1 : Représentation d'un problème de localisation générale

Les problèmes de localisation sont nombreux dans la littérature, nous allons donc nous concentrer sur ceux en rapport avec notre problème étudié, mais avant de les citer voici quelques définitions à prendre en compte :

I : Nombre d'usines.

J : L'ensemble d'entrepôts.

K : L'ensemble de clients.

C_{kj} : Le coût de transport d'une unité de local j au client k.

Z_{kj} : C'est une variable de décision qui indique si le client de la demande k est affecté à la ressource j. Si c'est le cas elle est égal à 1, sinon 0.

f_j : Le coût d'ouverture d'un local.

y_j : C'est une variable de décision qui indique si l'entrepôt j est ouvert ou non. Elle est égale à 1 si le local j est ouvert, ou 0 sinon.

S_j : La capacité maximale d'un local j.

d_k : Demande du client k.

1.2.1 Problème de localisation sans contrainte de capacité

Connue sous le nom de Uncapacited facility location problem (UFLP) : le problème de localisation sans contrainte de capacité est considéré comme l'un des problèmes de base dans la littérature des problèmes de localisation, car dans celui-ci on ne prend pas en compte des contraintes de capacité, ainsi chaque élément non sélectionné est alors attribué au plus proche sélectionné.

On représente l'UFLP par la formulation mathématique suivante [1] :

$$v(UFLP) = min\left(\sum_{k\in K}\sum_{j\in J} C_{kj}Z_{kj} + \sum_{j\in J} f_j y_j \right), \quad (1a)$$

$$s.t. \quad \sum_{j\in J} Z_{kj} = 1 \ \forall \mathrm{k} \in K, \quad (1b)$$

$$Z_{kj} - y_j \leq 0 \ \forall k \in K, j \in J, \quad (1c)$$

$$0 \leq Z_{kj} \leq 1, 0 \leq y_j \leq 1 \ \forall k \in K, j \in J, \quad (1d)$$

$$y_j \in \{0,1\} \ \forall \ j \in J. \quad (1e)$$

Dans ce modèle, (1a) représente la fonction objective où nous additionnons le coût d'un local et le coût d'allocation. Les contraintes (1b) garantissent que chaque local non sélectionné est attribué à un seul local sélectionné. Les contraintes (1c) garantissent que chaque local non sélectionné est attribué à un local sélectionné. Les contraintes (1d) et (1e) représentent les variables de décision.

1.2.2 Problème de localisation avec contrainte de capacité

Connue dans la littérature sous le nom de Capacited Facility Location Problem (CFLP), nous disposons dans ce problème d'un ensemble de locaux candidats et d'un ensemble de clients. Chaque client a une demande, chaque local a un coût d'installation et une capacité.

L'objectif est de choisir un sous-ensemble de locaux parmi un ensemble de candidats et d'affecter chaque client à un local sélectionné où la somme des demandes des clients alloués à un local sélectionné doit être inférieure ou égale à la capacité du local. Nous présentons ici le modèle mathématique qui décrit le CFLP [1] :

$$v(CFLP) = min\left(\sum_{k\in K}\sum_{j\in J} C_{kj}Z_{kj} + \sum_{j\in J} f_j y_j \right), \quad (2a)$$

$$s.t. \quad \sum_{j\in J} Z_{kj} = 1 \ \forall \mathrm{k} \in K, \quad (2b)$$

$$\sum_{k\in K} d_k Z_{kj} - S_j y_j \leq 0 \ \forall \mathrm{j} \in \mathrm{J}, \quad (2c)$$

$$Z_{kj} - y_j \leq 0 \ \forall k \in K, j \in J, \quad (2d)$$

$$\sum_{j\in J} S_j y_j \geq \mathrm{d}(K), \quad (2e)$$

$$\sum_{j\in J_q} Z_{kj} \leq 1 \ \forall k \in K, \forall q \in Q, \quad (2f)$$

$$0 \leq Z_{kj} \leq 1, 0 \leq y_j \leq 1 \ \forall k \in K, \forall j \in J, \quad (2g)$$

$$y_j \in \{0,1\} \ \forall j \in J. \quad (2h)$$

Dans ce modèle, (2a) représente la fonction objective où nous additionnons le coût d'un local et le coût d'allocation. Les contraintes (2b) garantissent que chaque local non sélectionné est attribué à un seul local sélectionné. Les contraintes (2c) garantissent que la capacité des locaux n'est pas dépassée. Les contraintes (2d) garantissent que chaque local non sélectionné est attribué à un local sélectionné. Les contraintes (2e) garantissent que la somme des capacités des locaux ouvertes doit être au moins égale à la demande totale des clients $\boldsymbol{d}(\boldsymbol{K})$. Les contraintes (2f) garantissent que les clients sont correctement assignés aux locaux. Les contraintes (2g) et (2h) représentent les variables de décision.

1.2.3 Problème du p-médiane

Le problème du p-médiane consiste à sélectionner, parmi un ensemble de $\boldsymbol{n}$ points de demandes donnés, un sous ensemble de $\boldsymbol{p}$ points d'approvisionnement et d'affecter les points (non sélectionnés) aux points sélectionnés. Dont l'objectif est de minimiser la somme des distances entre chaque point de demande et son point d'approvisionnement le plus proche [56].

1.2.4 Problème de couverture maximale

Le problème de couverture maximale (Maximal Covering Location Problem) consiste à sélectionner un ensemble de $\boldsymbol{m}$ locaux de manière à couvrir le plus grand nombre possible de clients. Un client est considéré comme couvert s'il se trouve à une distance égale ou inférieure à $\boldsymbol{r}$ du local le plus proche.

En d'autres termes, le but est de placer les locaux de manière stratégique pour maximiser la couverture des clients, ce qui se traduit par un nombre maximum de clients couverts [57].

1.2.5 Problème de localisation de hubs

Le problème de localisation de hubs (HLP) constitue une évolution récente et innovante des problèmes traditionnels de localisation des locaux. Il se focalise sur l'identification de points stratégiques, appelés hubs, qui agissent comme des centres de consolidation, de connexion et de commutation pour les flux logistiques entre diverses origines et destinations

Cette modalisation permet d'optimiser l'efficacité des réseaux de distribution en rationalisant les opérations de transport et en réduisant les coûts associés à la gestion des flux de marchandises ou de services [53].

1.2.6 Problèmes de localisation multi-produits

Dans la littérature, lorsque nous parlons d'un problème de localisation, dans la plupart du temps on part du principe que les clients ne demandent qu'un seul produit, alors qu'en réalité il est commun que les clients demandent plus d'un produit [1]. Dans ce cas, nous parlons d'une classe de problèmes de localisation appelée problèmes de localisation multi-produits.

1.2.7 Problèmes de localisation multi-niveaux

Un problème de localisation multi-niveaux est un type de problème de localisation où l'on retrouve un ou plusieurs intermédiaires entre les usines de production et les clients. Le but ici est toujours le même, c'est-à-dire choisir un sous-ensemble de locaux à installer à chaque niveau, afin de minimiser ou maximiser une fonction objective bien définie (comme le coût d'installation et des transports). Dans la figure 2, nous présentons un cas d'un problème de localisation multi-niveaux.

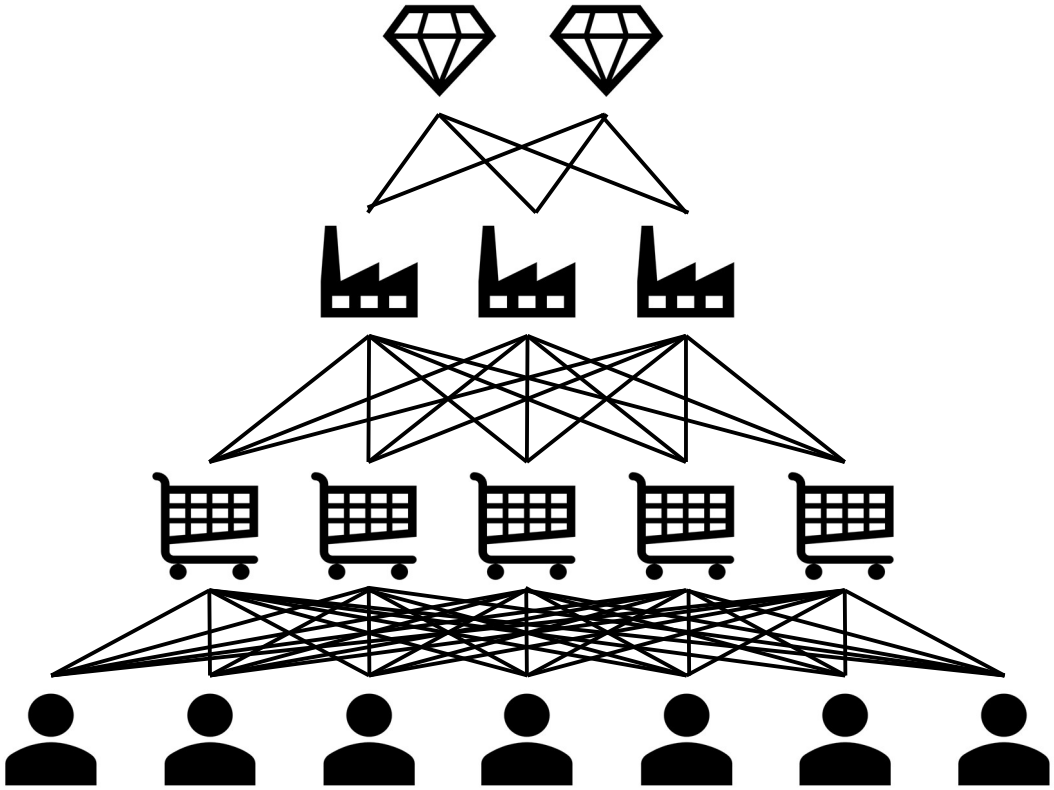


Figure 2 : Problème de localisation multi-niveaux

1.2.8 Problèmes de localisation multi-objectif

Les problèmes de localisation multi-objectif sont ceux qui visent à optimiser plusieurs critères en même temps (par exemple les coûts d'installation, distances et temps de livraison ... etc.). On peut retrouver dans la littérature plusieurs problèmes de localisation multi-objectif [4] [5] [6].

1.3 Problème de localisation à deux niveaux avec contrainte de capacité

Le problème de localisation à deux niveaux avec contrainte de capacité (Two-Stage Capacitated Facility Location Problem, TSCFLP) est une problématique complexe de localisation de locaux. Il s'agit de sélectionner un sous-ensemble d'entrepôts parmi les entrepôts candidats et un sous-ensemble d'usines parmi les usines disponibles pour satisfaire les demandes des clients, tout en minimisant les coûts totaux incluant les frais d'installation et d'allocation. Les entrepôts (généralement au niveau 1) et les usines (généralement au niveau 2) [7] doivent être attribués de manière à respecter leurs capacités, c'est-à-dire que la somme des demandes des clients pour un entrepôt sélectionné ne doit pas excéder sa capacité, de même pour les usines et les entrepôts qui leur sont attribués. Dans la littérature il y a plusieurs variantes du TSCFLP telles que le TSCFLP à source unique, le TSCFLP à sources multiples et le TSCFLP multi-produits. Dans ce mémoire, on se concentre spécifiquement sur le TSCFLP à sources multiples.

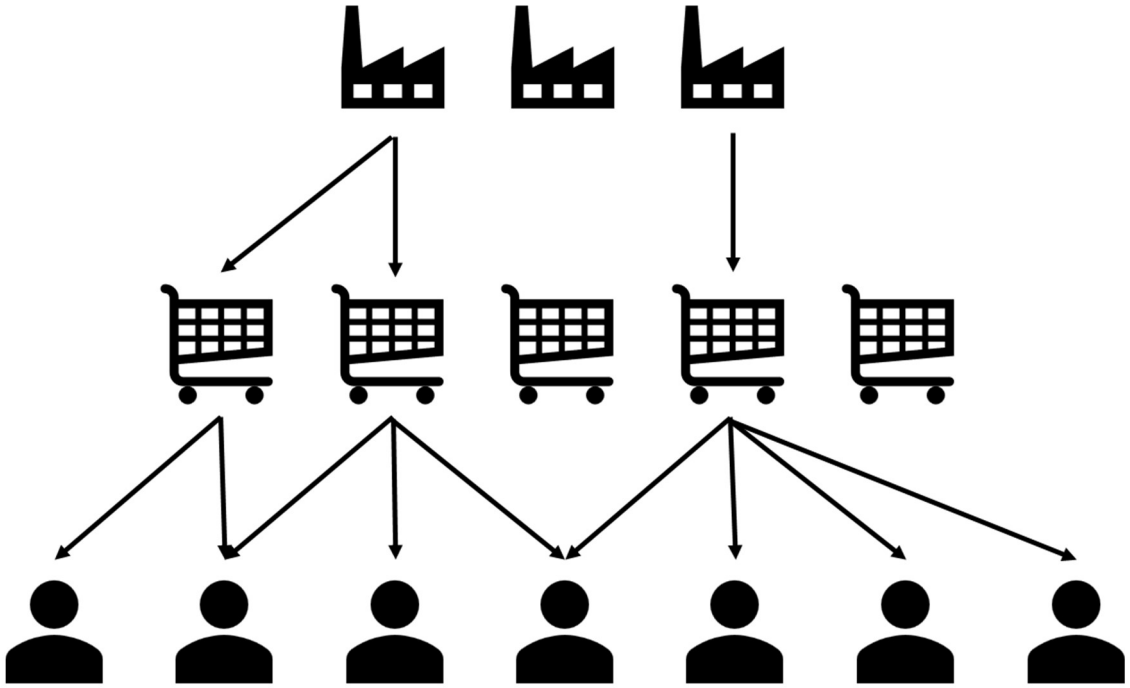


Figure 3 : Représentation d'un TSCFLP

1.4 Applications des problèmes de localisation

Les problèmes de localisation sont largement utilisés pour modéliser des problèmes industriels et concrets, démontrant ainsi leur utilité dans de nombreux domaines, parmi les plus connus [8] et [9] :

Analyse de clusters : Elle vise à regrouper des éléments de manière à ce que les clusters d'un même groupe soient similaires entre eux, tandis que ceux appartenant à des groupes différents présentent des différences significatives. La localisation implique ensuite de choisir des éléments représentatifs parmi l'ensemble global des éléments, tandis que l'allocation consiste à assigner les éléments restants aux clusters sélectionnés. À titre d'exemple, dans [10], les auteurs ont modélisé une tâche de regroupement sous forme de problème p-médiane.

Localisation des comptes bancaires : Lorsqu'une entreprise doit effectuer des paiements à ses fournisseurs, elle doit choisir les comptes bancaires à utiliser pour cette opération. En fonction de l'emplacement géographique des comptes utilisés, il est possible d'optimiser la gestion des fonds disponibles en le modélisant en un problème de localisation. Dans [11], les auteurs ont supposé que lorsqu'une entreprise paie ses fournisseurs, nous pouvons optimiser le délai de disponibilité des fonds en choisissant l'emplacement des comptes bancaires utilisés pour les paiements.

Sélection des fournisseurs : Chaque entreprise doit sélectionner des fournisseurs pour l'approvisionnement en produits. Cette sélection se base sur plusieurs critères comme le prix, la qualité, l'expertise…etc. Dans ce cas-là, la localisation implique de choisir certains fournisseurs parmi un ensemble donné. L'allocation concerne la décision de quel produit acheter auprès de quel fournisseur. Dans [12], les auteurs ont examiné comment le problème de la sélection des fournisseurs peut être formulé et résolu en utilisant des problèmes de localisation comme l'UFLP et le CFLP.

Localisation et dimensionnement des plates-formes offshore pour l'exploration pétrolière : Dans les articles [13] [14], les auteurs ont modélisé et résolu un problème d'exploitation pétrolière en tant que problème de localisation.

Localisation des bases de données dans les réseaux informatiques : Dans [15], les auteurs ont abordé le problème de l'installation et de la maintenance des bases de données dans un réseau informatique en le modélisant comme une variante étendue de l'UFLP.

Localisation des réseaux informatiques et des concentrateurs : Des études utilisent les problèmes de localisation pour résoudre divers problèmes complexes liés à la conception des réseaux de télécommunications et informatiques [16] et [17]. De plus, de nombreux de ces défis complexes sont spécifiquement liés à la localisation des concentrateurs [18] et [19].

Sélection d'index pour la conception de bases de données : La sélection des index pour la conception des bases de données est une étape importante. Chaque base de données se compose de tables contenant plusieurs ensembles de données. L'association des index à ces ensembles permet de stocker les entrées de manière triée, ce qui accélère les requêtes. Par exemple dans [20], les auteurs ont abordé un problème important dans la conception physique

des bases de données, à savoir le problème de sélection des index. Problème qui a été modélisé et résolu comme un UFLP.

1.5 Conclusion

Dans ce chapitre, nous avons vu quelques formes que le problème de localisation pouvait prendre, en nous concentrant sur ceux qui ressemblent le plus à notre problème « TSCFLP ». Puis nous avons présenté une description de notre problème « TSCFLP ». Enfin, on a exploré quelques-unes de ses applications dans la vie réelle et industrielle.

Chapitre 2 : Méthodes et algorithmes d'optimisation

2.1 Introduction

Dans ce chapitre, nous présenterons les différentes approches, techniques et algorithmes utilisés pour résoudre les problèmes d'optimisation. Ces méthodes peuvent être catégorisées en deux grandes classes : les méthodes exactes et les méthodes approchées. Fondamentalement, les méthodes exactes s'efforcent de trouver la solution optimale globale d'un problème. En revanche, les méthodes et les algorithmes approchées cherchent à obtenir une solution de bonne qualité (optimale ou proche de l'optimale) dans un temps de calcul raisonnable, car trouver la solution optimale pour un problème nécessite, en général, un temps de calcul exponentiel.

2.2 Définition générale d'un problème d'optimisation

En optimisation combinatoire, le but est de trouver la meilleure solution réalisable $\boldsymbol{s_b}$ parmi un ensemble fini, aussi appelé espace de recherche $\boldsymbol{S}$. Cela implique la minimisation ou la maximisation d'une fonction objective $\boldsymbol{f(s)}$, où $\boldsymbol{s} \in \boldsymbol{S}$.

La faisabilité d'une solution dépend de sa conformité aux contraintes du problème. En optimisation combinatoire, la plus grande difficulté est d'examiner toutes les solutions dans $\boldsymbol{S}$, surtout lorsque cet ensemble est vaste.

2.3 Méthodes approchées

Les problèmes d'optimisation sont souvent très complexes, avec de nombreuses contraintes, rendant les méthodes exactes inapplicables dans la plupart des cas. Ainsi, il est important de rechercher des solutions satisfaisantes dans un délai raisonnable plutôt que d'attendre une solution optimale nécessitant des années de calcul [21].

Contrairement aux algorithmes exacts, les méthodes approchées n'offrent pas de garantie d'optimalité, mais elles permettent de trouver des solutions de bonnes qualités en un temps réduit. Elles offrent un compromis entre la qualité de la solution et le temps de calcul. Dans la littérature, il existe de nombreuses méthodes approchées, classées en trois catégories : les heuristiques, les métaheuristiques et les méthodes hybrides.

2.3.1 Les Heuristiques

Dans la littérature, il existe plusieurs définitions d'une heuristique. Ici, nous exposons celle de Feigenbaum et Feldman (1963) [24] : *"Une heuristique (règle heuristique, méthode heuristique) consiste en une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre dispositif qui réduit considérablement l'exploration des solutions dans des espaces problématiques cruciaux. Les heuristiques ne promettent pas de solutions optimales. En fait, elles ne promettent pas de solution du tout. L'atout d'une heuristique utile réside dans sa capacité à fournir des solutions généralement satisfaisantes".*

Et donc, en optimisation combinatoire, une heuristique est considérée comme une méthode conçue pour résoudre un problème spécifique. Son objectif premier est de trouver des solutions de qualité (pas forcément optimales) dans un temps raisonnable [21].

1) Heuristique gloutonne de construction

Une heuristique gloutonne de construction [25] est une méthode qui construit progressivement une solution complète à partir d'une solution vide $\boldsymbol{S}$. À chaque itération, elle choisit à partir de la liste de candidat $\boldsymbol{C}$, le meilleur élément $\boldsymbol{e_b}$ selon une fonction d'évaluation, puis l'ajoute à la solution en cours de construction. Ce processus se répète jusqu'à ce qu'on obtienne une solution complète et réalisable. La fonction d'évaluation, également connue sous le nom de critère glouton ou règle de choix glouton, mesure généralement l'augmentation ou la diminution de la fonction objective quand un élément est ajouté à la solution partielle.

Algorithm 1: Heuristique gloutonne de construction pour un problème de minimisation

```
1  S ← ∅
2  C ← {e_1, e_2, ..., e_n}
3  Évaluer le coût incrémental c(e) pour tout e ∈ C
4  while C ≠ ∅ do
5  |   e_b ← sélectionner e ∈ C avec le plus petit coût incrémental c(e)
6  |   S ← S ∪ {e_b}
7  |   C ← C − {e_b}
8  |   foreach e ∈ C do
9  |   |   Réévaluer le coût incrémental c(e)
10 |   end
11 end
12 return S
```

2) Heuristique gloutonne randomisée de construction

La randomisation joue un rôle important dans la conception des algorithmes [25]. Elle permet d'introduire la diversité dans le processus de recherche, ce qui aide l'algorithme à explorer d'une manière efficace l'espace des solutions et à éviter de tomber dans des optimums locaux. Un usage particulièrement important de la randomisation se trouve dans les algorithmes gloutons.

Une heuristique gloutonne randomisée de construction [25] suit le même principe qu'un algorithme glouton classique, mais au lieu de choisir systématiquement le meilleur élément, il crée une liste restreinte de meilleurs éléments $\boldsymbol{e}$ parmi les éléments candidats de l'ensemble $\boldsymbol{C}$ et en sélectionne un au hasard. En général, les algorithmes gloutons randomisés sont utilisés lors de la phase de construction du GRASP ou pour générer des solutions initiales pour les algorithmes génétiques (GA).

Algorithm 2: Heuristique gloutonne randomisée de consruction pour un problème de minimisation

$S \leftarrow \emptyset$
$C \leftarrow \{e_1, e_2, \ldots, e_n\}$
Évaluer le coût incrémental $c(e)$ pour tout $e \in C$
while $C \neq \emptyset$ **do**
    Construire une liste avec les éléments candidats ayant les plus petits coûts incrémentaux
    $e_b \leftarrow$ Sélectionner aléatoirement $e \in$ liste restreinte des candidats
    $S \leftarrow S \cup \{e_b\}$
    $C \leftarrow C - \{e_b\}$
    Réévaluer le coût incrémental $c(e)$ pour tout $e \in C$
end
return S

3) Algorithme de recherche locale

Un algorithme de recherche locale (LS) améliore, de manière itérative, une solution. En la remplaçant par une meilleure solution de son voisinage. Ce voisinage est créé en effectuant de petites modifications à la solution actuelle. L'efficacité de cet algorithme dépend quand même de plusieurs facteurs, tels que la solution initiale et la structure du voisinage. L'algorithme s'arrête lorsqu'il n'y a plus de meilleure solution dans le voisinage.

Algorithm 3: Algorithme De Recherche Locale

```
S ← solution initiale
while S n'est pas un optimum local do
    S_n ← sélectionner S ∈ N(S)
    if f(S_n) est meilleur que f(S) then
        S ← S_n
    end
end
return S
```

2.3.2 Les métaheuristiques

D'après Osman et Laporte (1996) [26] : *" une métaheuristique représente un processus itératif qui supervise et guide une heuristique, en fusionnant judicieusement plusieurs concepts afin d'explorer et d'exploiter l'intégralité de l'espace de recherche. Des stratégies d'apprentissage sont employées pour structurer l'information en vue de trouver efficacement des solutions optimales, ou proches de l'optimalité".*

Une heuristique est un algorithme conçu pour résoudre un problème spécifique. En revanche, une métaheuristique constitue une approche générale pouvant être appliquée à une multitude de problèmes d'optimisation. Diverses métaheuristiques ont été élaborées, pouvant être classées en deux grandes catégories : les méthodes basées sur une solution unique et les méthodes basées sur une population de solutions.

a) Les méthodes basées sur une solution unique

Les méthodes basées sur une solution unique sont des algorithmes qui se concentrent sur une seule solution et explorent généralement son voisinage pour effectuer des améliorations. Après avoir créé une solution initiale, ces méthodes cherchent à améliorer progressivement la qualité de la solution en analysant et en explorant les solutions voisines.

Il existe dans la littérature plusieurs métaheuristiques basées sur une seule solution, parmi les méthodes les plus connues on cite : le recuit simulé, la recherche Tabou, la recherche à voisinage variable et la méthode GRASP.

1) Recuit Simulé

Le recuit simulé (SA), une méthode proposée par Kirkpatrick [27], qui s'inspire du processus de recuit en métallurgie, un traitement thermique visant à améliorer les propriétés d'un métal. Ce traitement implique le chauffage du métal à une température définie, suivie d'un refroidissement contrôlé pour permettre une réorganisation ordonnée des atomes, réduisant ainsi les imperfections durant le passage de l'état liquide à l'état solide.

Pour résoudre des problèmes d'optimisation, Cet algorithme commence avec une température $\boldsymbol{T}$ initialement très élevée et une solution initiale $\boldsymbol{S}$. Tout au long de l'algorithme, la température est progressivement réduite. À chaque itération, une nouvelle solution voisine $\boldsymbol{S_n}$ est générée aléatoirement et peut être acceptée si elle est meilleure que la solution actuelle. Si elle est moins bonne, elle peut tout de même être acceptée avec une probabilité qui diminue en fonction de la température. Ce processus permet à l'algorithme de refuser progressivement les mauvaises solutions et de converger vers des solutions de haute qualité $\boldsymbol{S_b}$.

Algorithm 4: Algorithme De Recuit Simulé

```
S ← Solution initiale;
S_b ← S;
T ← Température initiale;
while le critère d'arrêt n'est pas satisfait do
    Choisir aléatoirement une solution voisine S_n ∈ N(S);
    r ← un nombre aléatoire entre 0 et 1;
    Calculer Δ;
    if S_n est meilleure que S ou r < e^{-Δ/T} then
        S ← S_n;
        if S est meilleure que S_b then
            S_b ← S;
        end
    end
    Mettre à jour T;
end
return S_b;
```

2) Recherche Tabou

La recherche tabou (TS), développée par Glover en 1986 [28] [60], est une méthode de recherche qui utilise une liste tabou pour enregistrer les solutions déjà explorées, afin d'éviter de les revisiter immédiatement. Cette liste fonctionne comme un dispositif de mémoire, nous permettant de sortir des optimums locaux, et encourageant l'exploration de nouvelles zones de l'espace de recherche.

L'algorithme de recherche tabou commence par créer une solution initiale $\boldsymbol{S}$, qui deviendra par la suite la solution courante. Puis, à chaque itération, il crée le voisinage $\boldsymbol{N(S)}$ de la solution courante et sélectionne la meilleure solution $\boldsymbol{S_n}$ qui n'est pas présente dans la liste tabou $\boldsymbol{L}$. Ensuite, elle est mise à jour en y ajoutant la nouvelle solution sélectionnée et en retirant la plus ancienne selon un système FIFO (premier entré premier sorti). La solution choisie devient alors la nouvelle solution courante. L'algorithme conserve en mémoire la meilleure solution globale $\boldsymbol{S_b}$ rencontrée, et il s'arrête lorsque le critère de fin est atteint.

Algorithm 5: Algorithme De Recherche Tabou

```
S ←Solution initiale;
S_b ←S;
while le critère d'arrêt n'est pas satisfait do
    Créer N(S);
    S_n ← Choisir la meilleure solution s ∈ N(S), avec S_n ∉ L;
    Mettre à jour la liste L;
    S ← S_n;
    if S est meilleure que S_b then
        S_b ← S;
    end
end
return S_b;
```

3) Recherche à voisinage variable

La recherche à voisinage variable (Variable Neighbourhood Search, VNS) est une métaheuristique développée par Mladenović et Hansen en 1997 [30]. Cette technique repose sur un changement des structures de voisinage, associé à des perturbations de la solution initiale et à l'application de méthode de recherche locale. Au cours de l'application de l'algorithme, différents ensembles de voisinages prédéfinis $\boldsymbol{N(k)}$ sont explorés pour trouver de nouvelles solutions potentielles [55].

Les principes de cette méthode ainsi que ses applications ont été documentés et analysés dans diverses études et articles de synthèse, offrant ainsi plusieurs aperçus de la manière dont le VNS a été adapté pour résoudre les problèmes complexes dans divers domaines [31].

Algorithme 6 : Algorithme de recherche à voisinage variable

```
1  S ← solution initiale
2  définir un ensemble de structures de voisinage N(k), avec k ∈ [1 , k_max]
3  while le critère d'arrêt n'est pas satisfait do
4  |  k ← 1
5  |  while k < k_max do
6  |  |  S_1 ← Perturbation(S, N(k))
7  |  |  S_2 ← Recherche_Locale(S_1, N(k))
8  |  |  if f(S_2) < f(S) then
9  |  |  |  S ← S_2
10 |  |  |  k ← 1
11 |  |  else
12 |  |  |  k ← k + 1
13 |  |  end
14 |  end
15 end
16 return S
```

4) GRASP

La méthode GRASP (Greedy Randomized Adaptive Search Procedure) a été présentée en 1989 [29] comme une métaheuristique de démarrage multiple qui intègre à la fois une heuristique gloutonne randomisée et une recherche locale. Dans chaque itération, on commence par utiliser l'heuristique gloutonne randomisée pour trouver une solution. Si cette solution s'avère non réalisable, un mécanisme de réparation peut être activé pour ajuster ou générer une nouvelle solution faisable. Cette solution est ensuite améliorée par un algorithme de recherche locale. La meilleure solution obtenue au cours des différentes itérations est gardée comme le résultat final de l'algorithme.

Algorithme 7 : GRASP

```
1  f_i ← ∞;
2  while le critère d'arrêt n'est pas satisfait do
3  |  S ← Algorithme_Aléatoire_Glouton();
4  |  if (S n'est pas faisable) then
5  |  |  S ← Réparer_Solution(S);
6  |  end
7  |  S ← Recherche_Locale(S);
8  |  if (f(S) < f_i) then
9  |  |  S_b ← S;
10 |  |  f_i ← f(S);
11 |  end
12 end
13 retourner S_b;
```

b) Les méthodes basées sur une population de solutions

Les méthodes basées sur une population de solutions, sont connues pour leur utilisation d'une population de solutions pour explorer l'espace de recherche. Généralement inspirées de la nature [32], ces algorithmes commencent avec une population de solutions, et à chaque itération, cherchent à améliorer cette population en sélectionnant les solutions les plus prometteuses, et en appliquant des opérateurs de variation. Parmi les exemples les plus répandus de ces méthodes figurent : les algorithmes génétiques, l'algorithme d'optimisation par essaim de particules, l'algorithme de colonie de fourmis, l'algorithme de colonie d'abeilles artificielles et l'algorithme d'optimisation par des baleines.

1) Algorithmes génétiques

L'algorithme génétique (GA), proposé par Holland en 1975 [33], est inspiré de la théorie d'évolution dans la nature et des principes de sélection naturelle et génétique. Il repose principalement sur deux mécanismes : la sélection d'individus pour engendrer la génération suivante, et la manipulation de ces individus sélectionnés à travers des techniques de croisement et de mutation pour produire une nouvelle génération [34].

L'algorithme commence par la construction d'une population initiale de solutions. Ensuite, à chaque itération, il sélectionne un ensemble de solutions nommées parents en dupliquant certaines des solutions choisies de la population existante. Chaque solution peut être : non sélectionnée ou sélectionnée une ou plusieurs fois. Puis, un croisement est effectué sur les parents pour créer un ensemble de solutions enfants. Après le croissement, un opérateur de mutation est appliqué sur les solutions enfants obtenues. À la fin de chaque itération, l'algorithme sélectionne des solutions parmi la population actuelle et les enfants pour créer la population de l'itération suivante.

Algorithme 8 : Algorithme Génétique

```
1 P ← Créer_Population_Initiale();
2 while (le critère d'arrêt n'est pas satisfait) do
3 |   P_n ← Sélection(P);
4 |   E ← Croisement(P_n);
5 |   E ← Mutation(E);
6 |   P ← Remplacement(E, P);
7 end
8 retourner Meilleure solution trouvée;
```

i. Principes de la sélection naturelle

La sélection naturelle repose sur trois principes : la variation, l'adaptation et l'hérédité.

- **La variation :** Les individus d'une même espèce présentent des différences plus ou moins importantes au sein d'une population.
- **L'adaptation :** Certains individus possèdent des caractéristiques qui augmentent leurs chances de survie et de reproduction par rapport aux autres.
- **L'hérédité :** Les caractéristiques des individus sont transmises à leur descendance.

ii. Le croissement

Le croisement est un processus clé de l'algorithmique génétique, permettant de générer une ou plusieurs nouvelles solutions, nommées "enfants", par la recombinaison des gènes de deux "parents". Ce processus est appliqué à chaque couple de solutions sélectionnées, avec une probabilité de croisement $\boldsymbol{P_c}$, généralement située entre 65% et 90%. Il existe diverses techniques de croisement, telles que : le croisement à un point, le croisement à deux points et le croisement uniforme.

- **Croisement à un point**

Cette technique divise chaque parent en deux segments. Le lieu de séparation, ou point de découpage, est déterminé de manière aléatoire.

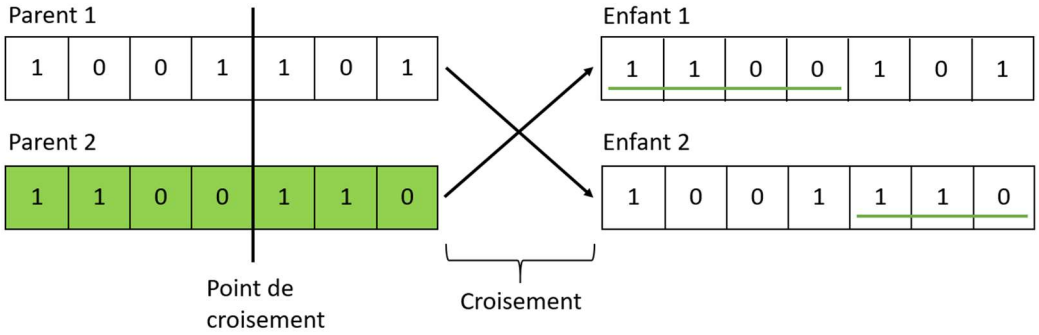


Figure 4 : Le croisement à un point dans l'algorithme génétique

- **Croisement à 2 points**

Cette approche divise chacun des deux parents en trois segments, en utilisant deux points de découpage. Ces deux points sont sélectionnés de façon aléatoire.

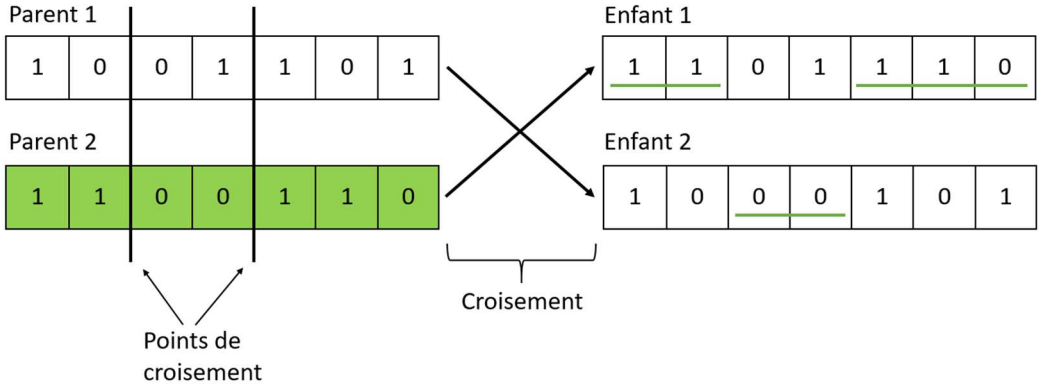


Figure 5 : Le croisement à deux points dans l'algorithme génétique

- **Croisement uniforme**

Cette technique implique d'examiner les deux parents gène par gène, et à chaque fois, un gène est choisi aléatoirement parmi les deux. Les gènes choisis sont ensuite assemblés pour former la nouvelle solution, appelée "enfant".

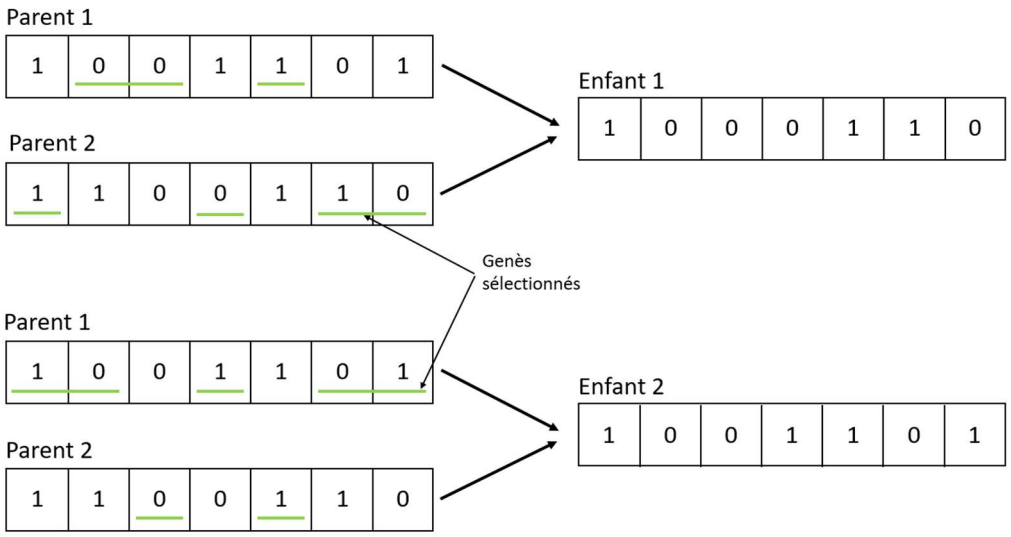


Figure 6 : Le croisement uniforme dans l'algorithme génétique

iii. La mutation

La mutation implique de réaliser un petit changement aléatoire dans une solution, comme la modification d'un ou deux gènes. L'objectif de la mutation est de favoriser une exploration efficace de l'espace de recherche. La probabilité d'appliquer une mutation, notée $\boldsymbol{P_m}$ varie généralement entre 1-5%.

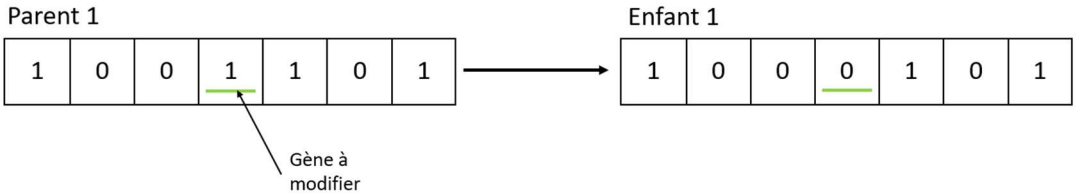


Figure 7 : La mutation dans l'algorithme génétique

iv. La sélection

L'opérateur de sélection est responsable de l'identification et de la sélection des meilleures solutions au sein de la population $\boldsymbol{P}$, afin de constituer l'ensemble des parents $\boldsymbol{P'}$. Ce dernier est formé en copiant les solutions sélectionnées. Dans la littérature, $\boldsymbol{P'}$ est souvent nommée *la population intermédiaire*. Chaque solution de $\boldsymbol{P}$ peut être : non sélectionnée, sélectionnée une ou plusieurs fois. Plusieurs méthodes de sélection sont utilisées, parmi les plus utilisée : la sélection par roulette, la sélection par rang, la sélection par tournoi, et la sélection uniforme.

- **Sélection par roulette**

Cette méthode consiste à sélectionner aléatoirement une solution, avec une probabilité relative à la valeur de sa fonction objective. La probabilité de sélection $\boldsymbol{Pr_s}$ d'une solution $\boldsymbol{s}$ est calculée comme suit :

$$\boldsymbol{Pr_s} = \frac{f(s)}{\sum f(s'), s' \in |P|}$$

Pour un problème de minimisation, après avoir calculé $\boldsymbol{Pr_s}$ on transforme la probabilité de sélection de chaque solution comme suit :

$$\boldsymbol{Pr_s} = \frac{1 - \boldsymbol{Pr_s}}{|P| - 1}$$

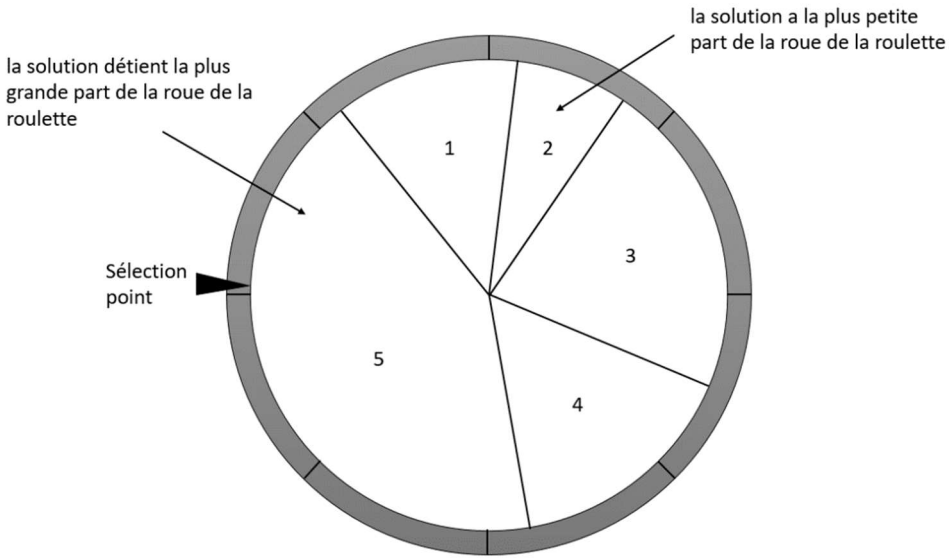


Figure 8 : La sélection par roulette dans l'algorithme génétique

- **Sélection par rang**

Avec cette méthode, chaque solution est choisie de manière aléatoire, avec une probabilité de sélection relative à son rang dans la population. Les étapes de cette méthode sont les suivantes :

La première étape consiste à trier les solutions de la population $\boldsymbol{P}$ en utilisant la fonction objective. Le tri commence par la solution la moins bonne ($rang = 1$) et se termine par la meilleure solution($rang = |\boldsymbol{P}|$, le nombre total de solutions dans la population).

Ensuite, la probabilité de sélection $\boldsymbol{Pr_s}$ de chaque solution $\boldsymbol{S}$ est calculée en utilisant la formule suivante :

$$\boldsymbol{Pr_s} = \frac{\boldsymbol{rang(s)}}{\sum \boldsymbol{rang(s')}, s' \in |P|}$$

Cette méthode garantit que les meilleures solutions ont une probabilité plus élevée d'être sélectionnées, tout en laissant une chance aux autres solutions d'être choisies.

- **Sélection par tournoi**

Cette méthode consiste à sélectionner aléatoirement $\boldsymbol{k}$ solutions ($\boldsymbol{k} < |\boldsymbol{P}|$) à partir de la population $\boldsymbol{P}$. Ensuite, la meilleure solution parmi les $\boldsymbol{k}$ solutions choisies est sélectionnée.

- **Sélection uniforme**

Cette méthode sélectionne aléatoirement une solution sans prendre en compte la valeur de la fonction objective. Par conséquent, chaque solution a une probabilité égale à $\mathbf{1/|P|}$ d'être choisie, où $|\boldsymbol{P}|$ représente la taille de la population.

v. Le remplacement

Il existe plusieurs méthodes pour sélectionner les solutions de la population pour l'itération suivante :

- **Remplacement total :** La population $\boldsymbol{P}$ est entièrement remplacé par la nouvelle population $\boldsymbol{E}$.
- **Fusion et sélection aléatoire :** Les deux ensembles $\boldsymbol{P}$ et $\boldsymbol{E}$ sont combinés, puis une méthode de sélection est utilisée.
- **Sélection élitiste :** Les ensembles $\boldsymbol{P}$ et $\boldsymbol{E}$ sont combinés, puis les meilleures solutions sont sélectionnées afin de créer la nouvelle population.

2) L'optimisation par essaim de particules

L'optimisation par essaim de particulaire (PSO) a été introduite en 1995 par Kennedy et Eberhart [35]. Inspirée par les mouvements groupés des oiseaux où chaque individu peut tirer parti des expériences de l'ensemble du groupe.

Dans cette métaheuristique, des particules se déplacent dans l'espace de solution en ajustant leur position, en fonction de leur propre meilleure position passée et de la meilleure position globale trouvée par le groupe. Chaque particule met à jour sa vitesse et sa position, ce qui permet à l'essaim de converger vers une solution optimale.

Algorithme 9 : Optimisation Par Essaim De Particules

```
Initialiser_Vitesse(particules);
Initialiser_Position(particules);
while le critère d'arrêt n'est pas satisfait do
    foreach particule p do
        Évaluer_Fitness(p);
        Mise_A_Jour_Vitesse(p);
        Mise_A_Jour_Position(p);
        Mise_A_JourMeilleur(p);
    end
    Mise_A_JourMeilleur(Particules);
end
return Meilleure solution trouvée;
```

3) Algorithmes inspirés des colonies de fourmis

L'Optimisation par Colonie de Fourmis (Ant Colony Optimization, ACO) a été initialement développée par Dorigo dans les années 90 [36], inspirée par les comportements des fourmis qui laissent des traces de phéromones pour naviguer vers leur nid, ou découvrir des sources de nourriture. Ces traces agissent comme un système de communication, permettant un échange d'informations indirect entre les fourmis.

L'ACO se base essentiellement sur la communication indirecte au sein d'une colonie composée d'agents simples, nommées fourmis artificielles, qui interagissent via des pistes de phéromones artificielles. Ces pistes de phéromones constituent une information distribuée, utilisée par les fourmis pour construire des solutions au problème de manière probabiliste, et sont ajustées durant l'exécution de l'algorithme pour refléter leur expérience de recherche.

Algorithme 10 : Optimisation par Colonie de Fourmis

```
Initialiser_pheromones()
while Critere d'Arret est verifié do
    foreach fourmi f do
        Contruction_De_Solution()
        Mise_A_Jour_Pheromones()
    end
end
return Meilleure solution trouvée
```

4) L'algorithme de colonie d'abeilles artificielles

L'algorithme de colonie d'abeilles artificielles et l'algorithme (Artificial Bee Colony, ABC), introduit par Karaboga en 2005 [51], est une métaheuristique qui s'inspire du comportement des abeilles. On retrouve dans cette méthode trois groupes d'abeilles :

- **Les abeilles employées** : Elles sont associées à des sources de nourriture spécifiques.
- **Les abeilles observatrices** : Elles regardent le comportement des abeilles employées pour choisir une source de nourriture.
- **Les abeilles éclaireuses** : Elles cherchent aléatoirement des sources de nourriture.

Initialement, toutes les positions des sources de nourriture sont découvertes par les abeilles éclaireuses. Ensuite, le nectar des sources de nourriture est exploité par les abeilles employées et les abeilles observatrices, et cette exploitation continue les épuisera finalement. L'abeille employée dont la source de nourriture a été épuisée devient une abeille éclaireuse.

Dans l'ABC, la position d'une source de nourriture représente une solution possible au problème, et la quantité de nectar d'une source de nourriture correspond à la qualité de la solution associée. Le nombre d'abeilles employées est égal au nombre de sources de nourriture (solutions), car chaque abeille employée est associée à une seule et unique source de nourriture.

Algorithme 11 : Algorithme de colonie d'abeilles artificielles

```
1 Initialiser les abeilles;
2 Memoriser la meilleure solution actuelle;
3 while (le critère d'arrêt n'est pas satisfait) do
4 |   Phase des abeilles employées;
5 |   Phase des abeilles observatrices;
6 |   Phase des abeilles éclaireuses;
7 |   S_b ← la meilleure solution trouvée jusqu'à présent;
8 end
9 retourner S_b;
```

5) L'algorithme d'optimisation par des baleines

L'algorithme d'optimisation par des baleines (Whale Optimization Algorithm, WOA) est une métaheuristique développé par Mirjalili et Lewis en 2016 [52]. Inspirée du comportement de chasse des baleines à bosse. Il commence par générer un ensemble de solutions aléatoires, puis met à jour les positions des agents de recherche en fonction soit de la meilleure solution trouvée, soit d'un agent de recherche choisi au hasard. Un paramètre $\boldsymbol{a}$ diminue progressivement de 2 à 0 pour équilibrer l'exploration et l'exploitation. En fonction de la valeur d'une variable $\boldsymbol{p}$, l'algorithme alterne entre des mouvements en spirale et des mouvements circulaires. Cette adaptabilité permet à l'algorithme WOA de converger vers de bonnes solutions en évitant les optimums locaux.

Algorithme 12 : Algorithme de l'optimisation par des baleines

```
Initialiser(Population);
S_b ← La meilleure solution initiale;
while Critère d'arrêt non satisfait do
    foreach baleines b do
        Mettre_à_jour_position(b) en fonction de la meilleure solution;
    end
    Appliquer la phase de migration;
    Mettre_à_jour(Population);
    S_b ← la meilleure solution s ∈ Population;
end
return S_b;
```

2.4 Méthodes Exactes

Les méthodes exactes essayent de trouver la solution optimale d'un problème en examinant de manière exhaustive toutes les solutions potentielles dans l'espace de recherche. Cependant, leur principal inconvénient réside dans le temps d'exécution qui augmente de manière exponentielle avec la taille de l'instance du problème. Par conséquent, ces techniques sont souvent inadaptées aux problèmes ayant de grand espace de recherche [21]. Parmi les méthodes exactes les plus connues, nous citons : la programmation dynamique, la génération de colonnes, l'algorithme A*, ainsi que des approches telles que branch & bound, branch & cut, branch & price, et autres.

Branch and bound

L'algorithme de branch and bound (B&B) est apparu pour la première fois dans les années 60 [22] pour résoudre des problèmes de programmation linéaire. Mais il est devenu par la suite connu pour résoudre des problèmes d'optimisation NP-difficiles [23].

L'objectif principal du B&B est d'examiner toutes les solutions possibles, en éliminant les branches inutiles, c'est-à-dire celles qui contiennent des solutions irréalisables ou de mauvaise qualité.

Pour développer une méthode B&B qui donne de bons résultats, on utilise les techniques suivantes.

- **La technique de séparation :** Il s'agit de subdiviser l'espace de recherche en sous-ensembles de solutions tout en s'assurant que l'union de ces sous-ensembles couvre toutes les solutions possibles du problème.
- **La technique d'évaluation :** C'est elle qui détermine s'il existe des solutions de bonne qualité dans une branche de l'arbre en y calculant les bornes inférieure et supérieure associées à la branche actuelle.

- **La technique d'exploration :** C'est une stratégie d'exploration de l'arbre : elle détermine l'ordre de visite des branches. Il en existe plusieurs : meilleure d'abord, en profondeur d'abord… etc.

Algorithme 13 : Branch and Bound pour un problème de minimisation

```
1  T_root ← Créer la racine de l'arbre de recherche selon la technique de
     séparation
2  U_bound ← +∞
3  L ← {T_root}
4  while L ≠ ∅ do
5  |  S_C ← Explorer(L)
6  |  if Évaluation(S_C) ≤ U_bound then
7  |  |  L' ← Toutes les solutions partielles S' obtenues à partir de S_C
8  |  |  foreach S' dans L' do
9  |  |  |  if S' est une solution complète then
10 |  |  |  |  mettre à jour U_bound
11 |  |  |  |  mettre à jour S_best
12 |  |  |  else
13 |  |  |  |  ajouter S' à L
14 |  |  |  end
15 |  |  end
16 |  else
17 |  |  supprimer S_C de L
18 |  end
19 end
20 return S_best
```

2.5 Méthodes hybrides

L'hybridation consiste à combiner plusieurs méthodes d'optimisation pour développer des méthodes plus efficaces. Les métaheuristiques hybrides sont des algorithmes plus adaptés pour les situations complexes et réelles [37]. Cette section introduira les approches d'hybridation, y compris la combinaison de métaheuristiques avec des méthodes exactes et la combinaison de métaheuristiques entre elles.

2.5.1 Combinaison des métaheuristiques avec des méthodes exactes

Initialement, l'hybridation s'est concentrée sur la collaboration avec diverses métaheuristiques [38]. Cependant, les chercheurs ont découvert que les méthodes exactes et les métaheuristiques fonctionnent bien ensemble. Les méthodes exactes sont plus efficaces

pour résoudre des problèmes de petite taille et évaluer l'optimalité d'une solution, mais elles ne sont pas couramment utilisées pour les grands problèmes NP-difficiles en raison de leur temps d'exécution. Par exemple, on peut utiliser une méthode exacte pour résoudre un sous-problème au sein d'une méthode approchée, ou utiliser une méthode approchée pour calculer une borne inferieure et/ou supérieure pour une méthode exacte [21].

2.5.2 Combinaison des métaheuristiques avec d'autres métaheuristiques

La combinaison entre métaheuristiques est le type d'hybridation le plus répandu dans la littérature [38]. La fusion des métaheuristiques améliore les performances globales dans la résolution des problèmes d'optimisation. Les métaheuristiques peuvent être combinées de diverses manières. Voici plusieurs méthodes couramment utilisées [40], [41], [42] :

a) Les hybrides parallèles

Les hybrides parallèles impliquent plusieurs métaheuristiques fonctionnant simultanément. Les métaheuristiques travaillent indépendamment pour explorer l'espace de recherche et générer des solutions en même temps. Ces solutions sont ensuite comparées pour choisir la meilleure. La parallélisation est généralement utilisée pour améliorer la vitesse de recherche et la qualité des solutions pour des problèmes à grande échelle [40].

Les auteurs de [41] ont constaté que les stratégies de parallélisation pour les métaheuristiques diffèrent selon qu'elles sont basées sur des solutions uniques ou sur des populations de solution. Trois méthodes de parallélisation sont couramment utilisées pour les métaheuristiques basées sur des solutions uniques : « ***Parallel moves model »*** [42], « ***Parallel multi-start model*** » [43] et « ***Move acceleration model »*** [44].

De plus, les auteurs de [58] ont classé les hybrides parallèles en deux catégories :

- **Les hybrides parallèles synchrones :** où un algorithme de recherche est utilisé à la place d'un opérateur. Par exemple, on peut remplacer l'opérateur de mutation d'un algorithme génétique par une recherche tabou.
- **Les hybrides parallèles asynchrones :** où plusieurs algorithmes de recherche travaillent en même temps et s'échangent des informations.

b) Les hybrides séquentiels :

Les hybrides séquentiels, sont des algorithmes où deux méthodes s'exécutent l'une après l'autre et les résultats de la première sont les solutions initiales de la deuxième.

2.6 Conclusion

Dans ce chapitre, nous avons commencé par une vision générale sur les problèmes d'optimisation. Ensuite, nous avons présenté les méthodes approchées. Après, nous avons exploré les heuristiques et cité quelques méthodes. Puis, nous avons défini le concept d'une métaheuristique, en présentant les plus utilisés dans la littérature. Après, nous avons abordé les méthodes exactes. Enfin, nous avons discuté du concept d'hybridation, qui consiste à combiner différentes méthodes d'optimisation.

Chapitre 3 : Une méthode de recherche tabou probabiliste pour résoudre le TSCFLP

3.1 Introduction

Dans ce chapitre, nous proposons un algorithme de recherche tabou probabiliste pour résoudre le problème de localisation à deux étages avec contrainte de capacité, un seul produit et plusieurs sources (TSCFLP). Cet algorithme consiste à générer une solution aléatoire et vise à l'améliorer pour obtenir une solution de meilleure qualité dans un délai raisonnable. Nous commencerons alors par présenter la définition mathématique du TSCFLP selon [47], puis nous présenterons les travaux de la littérature qui ont abordé le TSCFLP. Enfin, nous présenterons en détail l'algorithme que nous avons proposé pour résoudre ce problème, avant de finir avec une conclusion.

3.2 Définition du problème

Le problème de localisation à deux étages avec capacité de transport (TSCFL) consiste à sélectionner un sous-ensemble d'entrepôts $\overline{\boldsymbol{E}} \subseteq \boldsymbol{E}$ et d'usines $\overline{\boldsymbol{U}} \subseteq \boldsymbol{U}$ afin de satisfaire les demandes de tous les clients, tout en minimisant les coûts totaux d'ouverture et de transport.

Nous désignons l'ensemble des clients par $\boldsymbol{C}$, où chaque client $\boldsymbol{c} \in \boldsymbol{C}$ possède une demande $\boldsymbol{q_c}$ à satisfaire. $\boldsymbol{E}$ Représente les entrepôts, pour chaque entrepôt $\boldsymbol{e} \in \boldsymbol{E}$ il existe une capacité $\boldsymbol{p_e}$, un coût d'ouverture $\boldsymbol{O_e}$, et un coût de transport de produit $\boldsymbol{d_{ec}}$ vers tous les clients $\boldsymbol{c} \in \boldsymbol{C}$. Et pour les usines, elles sont représentées par $\boldsymbol{U}$, chaque usine $\boldsymbol{u} \in \boldsymbol{U}$ est caractérisée par une capacité $\boldsymbol{p_u}$, un coût d'ouverture $\boldsymbol{O_u}$, et un coût de transport de produit $\boldsymbol{d_{ue}}$ vers tous les entrepôts $\boldsymbol{e} \in \boldsymbol{E}$.

Afin de bien représenter le modèle mathématique de notre problème, on va utiliser des variables de décisions spécifiques. Pour cela, nous introduisons les variables $\boldsymbol{a_c}$ pour chaque entrepôt $\boldsymbol{e} \in \boldsymbol{E}$. Et $\boldsymbol{b_u}$ pour chaque usine, $\boldsymbol{u} \in \boldsymbol{U}$, qui indiquent si ces locaux seront ouverts ou non. De plus, nous utilisons les variables de décision $\boldsymbol{v_{ue}}$ pour chaque paire usine-entrepôt ($\boldsymbol{u} \in \boldsymbol{U}, \boldsymbol{e} \in \boldsymbol{E}$) pour représenter la quantité de produit envoyée de l'usine $\boldsymbol{u}$ à l'entrepôt $\boldsymbol{e}$, et $\boldsymbol{w_{ec}}$ pour chaque paire entrepôt-client ($\boldsymbol{e} \in \boldsymbol{E}, \boldsymbol{c} \in \boldsymbol{C}$) pour indiquer la quantité de produit envoyée de l'entrepôt $\boldsymbol{e}$ au client $\boldsymbol{c}$. En fin, nous présentons le modèle mathématique du problème TSCFL tel qu'il est défini dans [47] :

$$min\left(\sum_{u\in U} O_u b_u + \sum_{e\in E} O_e\, a_e + \sum_{u\in U}\sum_{e\in E} d_{ue}\, v_{ue} + \sum_{c\in C}\sum_{e\in E} d_{ec}\, w_{ec}\right) \quad (3a)$$

$$s.t \quad \sum_{e\in E} w_{ec} \geq q_c \quad \forall\, c \in C \quad (3b)$$

$$\sum_{u\in U} v_{ue} \geq \sum_{c\in C} w_{ec} \quad \forall\, e \in E \quad (3c)$$

$$\sum_{e\in E} v_{ue} \leq p_u b_u \quad \forall\, u \in U \quad (3d)$$

$$\sum_{c\in C} w_{ec} \leq p_e a_e \quad \forall\, e \in E \quad (3e)$$

$$v_{ue} \in \mathbb{R}^+ \quad \forall\, u \in U, e \in E \quad (3f)$$

$$w_{ec} \in \mathbb{R}^+ \quad \forall\, e \in E, c \in C \quad (3g)$$

$$b_u \in \{0,1\} \quad \forall\, u \in U \quad (3h)$$

$$a_e \in \{0,1\} \quad \forall\, e \in E \quad (3i)$$

La fonction objective (3a) représente le coût total y compris le coût d'installation et le coût de transport. Les contraintes (3b) garantissent que chaque client est satisfait. Les contraintes (3c) sont des contraintes pour assurer que la quantité totale de produits expédiés depuis un entrepôt est au plus égale à la quantité totale expédiée vers celui-ci depuis les usines. Les contraintes (3d) et (3e) sont des contraintes de capacité des usines et des entrepôts, respectivement. Enfin, les contraintes (3f) et (3g) sont assignées aux variables de flux, et les contraintes (3h) et (3i) imposent des valeurs binaires pour les variables de décisions respectives.

3.3 Revue de la littérature

Dans cette partie nous présenterons les travaux de la littérature qui ont abordé le problème TSCFLP :

En 2014, Fernandes [47] ont proposé un algorithme génétique pour résoudre le TSCFLP. Pour tester l'algorithme, les auteurs ont créé une série d'instances différentes les unes des autres, et les résultats d'expérimentation sont ensuite comparés aux résultats obtenus par deux heuristiques lagrangiennes.

Puis en 2016, Louzada [48] ont développé une méthode hybride en combinant une recherche par regroupement (CS) pour définir les sites d'implantation des usines et des entrepôts, et une méthode exacte pour déterminer les flux de produits entre les usines, les entrepôts et les clients.

Après, en 2017 Guo [59] ont développé un algorithme évolutif hybride (HEA) qui utilise une méthode d'apprentissage automatique (appelée "extreme machine learning") pour résoudre le TSCFL.

Ensuite, en 2019 González [49] ont présenté une technique hybride qui combine la métaheuristique GRASP avec une procédure de “Local Branching”. Cette approche a donné des résultats significatifs, démontrant ainsi l'efficacité de cette méthode hybride.

En 2019 aussi, Biajoli [50] ont développé un algorithme génétique en utilisant une méthode de clés aléatoires biaisées (BRKGA) pour résoudre le même TSCFLP.

Et en 2021, González [7] ont combiné le Clustering Search (CS), la métaheuristique Adaptive Large Neighborhood Search (ALNS) et la technique de Local Branching (LB) pour le TSCFLP. Une hybridation qui a abouti à des solutions de meilleure qualité, obtenues en un temps de calcul nettement réduit.

Enfin, en 2022 Abboud [54] ont créé un algorithme de recuit simulé (SA) pour résoudre le problème de TSCFL.

3.4 Algorithme proposé pour résoudre le TSCFLP

Pour résoudre ce TSCFLP nous avons développé une recherche tabou probabiliste. Pour se faire, nous commençons par créer une solution aléatoire réalisable $\boldsymbol{S_0}$, qui deviendra par la suite la solution courante $\boldsymbol{S_c}$. Ensuite, nous créons la structure de voisinage $\boldsymbol{N(S_c)}$ de la solution $\boldsymbol{S_c}$ dont $\boldsymbol{N(S_c)}$ ne contient pas des solutions qui appartient à la liste tabou $\boldsymbol{L}$. Après la création de la structure de voisinage, nous cherchons la meilleure solution voisine $\boldsymbol{S_n}$ dans $\boldsymbol{N(S_c)}$, si $\boldsymbol{S_n}$ est meilleure que $\boldsymbol{S_c}$, alors $\boldsymbol{S_n}$ devient la solution courante. Sinon, on génère un nombre aléatoire $\boldsymbol{r}$ entre $\boldsymbol{0}$ et $\boldsymbol{P}$ pour sélectionner aléatoirement la $\boldsymbol{r^{ième}}$ meilleure solution voisine et cette solution devient la solution courante. Enfin, on met à jour la liste tabou et on vérifie si la solution courante $\boldsymbol{S_c}$ est meilleure que $\boldsymbol{S_b}$, et si c’est le cas, alors on met à jour $\boldsymbol{S_b}$. Par la suite, nous allons présenter en détails : (1) La procédure qui génère la solution initiale, (2) La procédure qui crée la structure de voisinage d’une solution, la procédure d’allocation (3), et enfin, nous expliquerons la procédure de mise à jour de la liste tabou (4).

Algorithme 14 : Algorithme De Recherche Tabou Probabiliste

```
1  S_0 ← génération d'une solution initiale réalisable ()
2  S_c ← S_0
3  S_b ← S_0
4  Inserer S_c dans la liste Tabou L
5  while le critère d'arrêt n'est pas satisfait do
6  |   CreerStructureDeVoisinage(N(s), L)
7  |   S_n ← Meilleur Voisin de N(S_c)
8  |   if S_n est meilleur que S_c then
9  |   |   S_c ← S_n
10 |   else
11 |   |   r ← nombre aléatoire entre 0 et p
12 |   |   S_c ← la r^ième meilleure solution voisine dans N(s)
13 |   end
14 |   Inserer S_c à la liste Tabou L
15 |   if S_c est meilleur que S_b then
16 |   |   S_b ← S_c
17 |   end
18 end
19 return S_b
```

3.4.1 Représentation de la solution

Nous représenterons notre solution par une structure de donnés composé de 4 vecteurs d'entier, contenant les Id des entrepôts et usines comme suit :

- Solution.Entrepôts_Ouverts : les Id des entrepôts ouverts de la solution.
- Solution.Entrepôts_Fermés : les Id des entrepôts fermés de la solution.
- Solution.Usines_Ouvertes : les Id des usines ouvertes de la solution.
- Solution.Usines_Fermées : les Id des usines fermées de la solution.

3.4.2 Procédure de génération d'une solution initiale

Cette procédure génère une solution de départ réalisable, elle choisit aléatoirement des entrepôts et des usines à ouvrir, jusqu'à ce que leur capacité totale puisse satisfaire toutes les demandes des clients. Après le choix des entrepôts et des usines, nous utilisons la procédure d'Allocation décrite ci-dessous pour allouer les clients aux entrepôts ouverts et les entrepôts ouverts aux usines ouvertes afin d'obtenir une solution complète réalisable.

Algorithme 15 : Génération d'une solution aléatoire realisable

```
1  Clients ← {c_1, c_2, ..., c_n};
2  Entrepots ← {e_1, e_2, ..., e_n};
3  Usines ← {u_1, u_2, ..., u_n};
4  Solution.Entrepots_Fermes ← Entrepots;
5  Solution.Usines_Fermees ← Usines;
6  Solution.Entrepots_Ouverts ← ∅;
7  Solution.Usines_Ouvertes ← ∅;
8  while (Les clients ne sont pas tous satisfais) do
9  |   e' ← selectionner aleatoirement un entrepot e ∈
   |    Solution.Entrepots_Fermes ;
10 |   Solution.Entrepots_Ouverts ← Solution.Entrepots_Ouverts ∪ {e'};
11 |   Solution.Entrepots_Fermes ← Solution.Entrepots_Fermes − {e'};
12 end
13 while (Les clients ne sont pas tous satisfais) do
14 |   u' ← selectionner aleatoirement une usine u ∈
   |    Solution.Usines_Fermees ;
15 |   Solution.Usines_Ouvertes ← Solution.Usines_Ouvertes ∪ {u'};
16 |   Solution.Usines_Fermees ← Solution.Usines_Fermees − {u'};
17 end
18 Procedure_Allocation(Solution, Clients);
19 return (Solution);
```

3.4.3 Procédure de voisinage

Dans la méthode de PTS proposée, nous construisons la structure de voisinage $\boldsymbol{N(S)}$ pour une solution $\boldsymbol{S}$ ($\boldsymbol{S_t}$ étant une solution temporaire), en utilisant 3 mouvements qui modifient la solution comme suit :

a) Swap : Ce mouvement consiste à créer un ensemble de solutions voisines en utilisant deux fonctions de permutation, $permuter1(Solution, e1, e2)$ qui échange un entrepôt ouvert ($e1 \in Solution.Entrepôts\_Ouverts$) avec un entrepôt fermé ($e2 \in Solution.Entrepôts\_Fermés$). Puis $permuter2(Solution, u1, u2)$ qui remplace une usine ouverte ($u1 \in Solution.Usines\_Ouvertes$) avec une autre fermée ($u2 \in Solution.Usines\_Fermées$). En s'assurant à chaque fois que cet échange satisfait les demandes des clients.

Algorithme 16 : Swap(Solution,Clients)

```
1  Solutions_voisines ← ∅
2  foreach Entrepôt e1 de Solution.Entrepôts_Ouverts do
3      foreach Entrepôt e2 de Solution.Entrepôts_Fermés do
4          S_t ← permuter1(Solution, e1, e2)
5          if S_t satisfait les demandes clients then
6              Procedure_Allocation(S_t, Clients)
7              Solutions_voisines ← Solutions_voisines ∪ {S_t}
8          end
9      end
10 end
11 foreach Usine u1 de Solution.Usines_Ouvertes do
12     foreach Usine u2 de Solution.Usines_Fermées do
13         S_t ← permuter2(Solution, u1, u2)
14         if S_t satisfait les demandes clients then
15             Procedure_Allocation(S_t, Clients)
16             Solutions_voisines ← Solutions_voisines ∪ {S_t}
17         end
18     end
19 end
20 return Solutions_voisines
```

b) Add : Ce mouvement consiste à créer un ensemble de solutions voisines en utilisant deux fonctions ajouter. $ajouter1(Solution, e)$ qui ajoute l'entrepôt $\boldsymbol{e}$ à $Solution.Entrepôts\_Ouverts$ ($\boldsymbol{e} \in Solution.Entrepôts\_Fermés$) et supprime $\boldsymbol{e}$ de $Solution.Entrepôts\_Fermés$. Puis, $ajouter2(Solution, u)$ qui ajoute l'usine $\boldsymbol{u}$ a $Solution.Usines\_Ouvertes$ ($\boldsymbol{u} \in Solution.Usines\_Fermées$) et supprime $\boldsymbol{u}$ de $Solution.Usines\_Fermées$.

Algorithme 17 : Add(Solution,Clients)

```
1  Solutions_voisines ← ∅
2  foreach Entrepôt e de Solution.Entrepôts_Fermés do
3  |   S_t ← Ajouter1(Solution, e)
4  |   Procedure_Allocation(S_t, Clients)
5  |   Solutions_voisines ← Solutions_voisines ∪ {S_t}
6  end
7  foreach Usine u de Solution.Usines_Fermées do
8  |   S_t ← Ajouter2(Solution, u)
9  |   Procedure_Allocation(S_t, Clients)
10 |   Solutions_voisines ← Solutions_voisines ∪ {S_t}
11 end
12 return Solutions_voisines
```

c) Drop : Ce mouvement consiste à créer un ensemble de solutions voisines en utilisant deux fonctions supprimer. $supprimer1(Solution, e)$ qui supprime l'entrepôt $\boldsymbol{e}$ de $Solution.Entrepôts\_Ouverts$ et l'ajoute à $Solution.Entrepôts\_Fermés$. Puis, $supprimer2(Solution, u)$ qui supprime l'usine $\boldsymbol{u}$ de $Solution.Usines\_Ouvertes$ et l'ajoute à $Solution.Usines\_Fermées$. En vérifiant à chaque fois que cette suppression satisfait les demandes des clients.

Algorithme 18 : Drop(Solution,Clients)

```
1  Solutions_voisines ← ∅
2  foreach Entrepôt e de Solution.Entrepôts_Ouverts do
3  |   S_t ← Supprimer1(Solution, e)
4  |   if S_t satisfait les demandes clients then
5  |   |   Procedure_Allocation(S_t, Clients)
6  |   |   Solutions_voisines ← Solutions_voisines ∪ {S_t}
7  |   end
8  end
9  foreach Usine u de Solution.Usines_Ouvertes do
10 |   S_t ← Supprimer2(Solution, u)
11 |   if S_t satisfait les demandes clients then
12 |   |   Procedure_Allocation(S_t, Clients)
13 |   |   Solutions_voisines ← Solutions_voisines ∪ {S_t}
14 |   end
15 end
16 return Solutions_voisines
```

Ensuite, on regroupe les 3 ensembles de solutions créées par les trois mouvements Swap, Add et Drop, puis on les ordonne en fonction de leur qualité. Enfin, on crée un ensemble $\boldsymbol{V}$ contenant uniquement les solutions non taboues (voir algorithme 19).

Algorithme 19 : Procèdure CreerStructureDeVoisinage(N(s), L)

$V \leftarrow \emptyset$
$Solutions\_Voisines \leftarrow \emptyset$
$Solutions\_Voisines \leftarrow Solutions\_Voisines \cup Swap(Solution, Clients)$
$Solutions\_Voisines \leftarrow Solutions\_Voisines \cup Add(Solution, Clients)$
$Solutions\_Voisines \leftarrow Solutions\_Voisines \cup Drop(Solution, Clients)$
$Solutions\_Voisines \leftarrow Ordonner(Solutions\_Voisines)$
foreach *solution* S_n *de* $Solutions\_Voisines$ **do**
    if $S_n \notin$ *la liste tabou* L **then**
        $V \leftarrow V \cup \{S_n\}$
    end
end
return V

3.4.4 Procédure d'allocation

Cette procédure a pour but d'attribuer à chaque client l'entrepôt (ou les entrepôts) ouvert le plus proche de lui, et pour chaque entrepôt ouvert, l'usine (les usines) ouverte la plus proche. Tout d'abord, les entrepôts sont classés en fonction de leur distance par rapport aux clients, puis nous classons les usines en fonction de leur distance par rapport aux entrepôts. Une fois ces deux classements établis, le meilleur entrepôt ouvert ***Entrepôt***, ayant une capacité non nulle de produits, est choisi pour répondre aux demandes du premier client insatisfaits et ce jusqu'à épuisement de sa capacité, il essaie de satisfaire complètement sa demande, ou à défaut, partiellement. Si la demande du client est satisfaite partiellement, sa demande est mise à jour et il reste un client insatisfait ; sinon, il est considéré comme satisfait et on passe aux clients suivants. Puis nous faisons de même pour les usines avec les entrepôts. La procédure se termine après que tous les clients et tous les entrepôts soient satisfaits.

Algorithme 20 : Algorithme d'allocation

```
1  Entrepôts_Ordonné ← Ordonner(Entrepôts)
2  Usines_Ordonné ← Ordonner(Usines)
3  foreach c ∈ Clients do
4      while c n'est pas satisfait do
5          Entrepot ← Le meilleur entrepot ouvert e ∈ Entrepôts_Ordonné,
            avec e.stock ≠ 0
6          if c.demande ≤ Entrepot.stock then
7              Entrepot.stock ← Entrepot.stock − c.demande
8              c.demande ← 0
9          else
10             c.demande ← c.demande − Entrepot.stock
11             Entrepot.stock ← 0
12         end
13     end
14 end
15 foreach e ∈ Entrepot do
16     while e n'est pas satisfait do
17         Usine ← La meilleur Usine ouverte u ∈ Usines_Ordonné, avec
            u.stock ≠ 0
18         if e.demande ≤ Usine.stock then
19             Usine.stock ← Usine.stock − e.demande
20             e.demande ← 0
21         else
22             e.demande ← e.demande − Usine.stock
23             Usine.stock ← 0
24         end
25     end
26 end
```

3.4.5 Gestion de la liste tabou

La liste tabou enregistre les solutions déjà explorées afin d'éviter le cyclique. À chaque itération, la solution courante est ajoutée à la liste tabou, et les solutions figurant dans cette liste ne peuvent pas être considérées comme candidates pour remplacer la solution courante. La liste tabou évolue tout au long de l'exécution de l'algorithme en se mettant à jour par un système FIFO. Autrement dit, quand la liste se remplie, la première solution enregistrée est effacée pour faire de la place à la solution courante.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une formulation mathématique de notre problème TSCFLP. Ensuite, nous avons présenté les travaux les plus proches de la littérature qui ont abordé ce problème. Après, nous avons proposé un algorithme de recherche tabou probabiliste pour résoudre ce problème en expliquant en détails les composants de cet algorithme : la structure d'une solution, la génération de solution aléatoire, l'exploration du voisinage, l'allocation des entrepôts et usines et enfin la gestion de la liste tabou.

Dans le chapitre 4, nous allons tester l'efficacité de l'algorithme proposé sur un jeu de données standard de la littérature.

Chapitre 4 : Tests et comparaison avec la littérature

4.1 Introduction

Dans ce chapitre, nous allons évaluer l'efficacité de l'algorithme que nous avons proposé à travers diverses expériences. Nous allons commencer par une description détaillée de jeu de données proposés par [47] et de la méthode utilisée pour leur génération. Ensuite, nous allons présenter un exemple parmi les instances. Enfin, nous présenterons les résultats obtenus par l'algorithme développé et nous comparerons ces résultats avec ceux des méthodes de la littérature.

4.2 Description du jeu de données de la littérature

Dans [47], les auteurs ont généré 50 instances pour le TSCFLP en utilisant les paramètres suivants :

- Nombre d'usines $I = 50$ ou 100.
- Nombre d'entrepôts $= 2 \times I$.
- Nombre de clients $= 4 \times I$.

Paramètres	b_i	f_i	c_{ij}	p_j	g_j	d_{jk}	q_k
Classe 1	$[2B\ 5B]$	$[2 \times 10^4\ \ 3 \times 10^4]$	[35 45]	$[2P\ 5P]$	$[8 \times 10^3\ \ 1.2 \times 10^4]$	[55 65]	[10 20]
Classe 2	$[5B\ 10B]$	$[2 \times 10^4\ \ 3 \times 10^4]$	[35 45]	$[5P\ 10P]$	$[8 \times 10^3\ \ 1.2 \times 10^4]$	[55 65]	[10 20]
Classe 3	$[15B\ 25B]$	$[2 \times 10^4\ \ 3 \times 10^4]$	[35 45]	$[15P\ 25P]$	$[8 \times 10^3\ \ 1.2 \times 10^4]$	$[8 \times 10^2\ \ 1 \times 10^3]$	[10 20]
Classe 4	$[5B\ 10B]$	$[2 \times 10^4\ \ 3 \times 10^4]$	$[50\ 1 \times 10^2]$	$[5P\ 10P]$	$[8 \times 10^3\ \ 1.2 \times 10^4]$	$[50\ 1 \times 10^2]$	[10 20]
Classe 5	$[5B\ 10B]$	$[2 \times 10^4\ \ 3 \times 10^4]$	[35 45]	$[5P\ 10P]$	$[8 \times 10^3\ \ 1.2 \times 10^4]$	$[8 \times 10^2\ \ 1 \times 10^3]$	[10 20]

Tableau 1 : Paramètres utilisés pour la génération des instances

Sachent que $B = \frac{\sum_{k \in K} q_k}{I}$ et $P = \frac{\sum_{k \in K} q_k}{J}$.

On obtient alors deux catégories d'instances, la ***première*** comprend 25 instances réparties en cinq classes d'instances avec $I = 50$. La ***deuxième*** est aussi composée de 25 instances réparties en cinq classes d'instances avec $I = 100$. Toutes les instances sont disponibles à l'adresse suivante : https://github.com/pehgonzalez/OCA.

Voici quelques captures d'écrans de l'instance PSC4-C2-50, de 200 clients, 100 entrepôts et 50 usines :

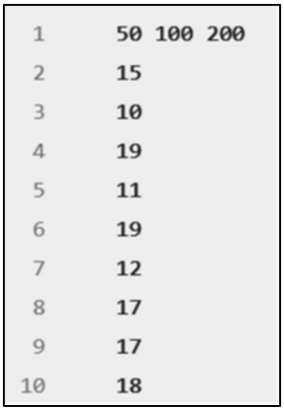


Figure 9 : Partie clients

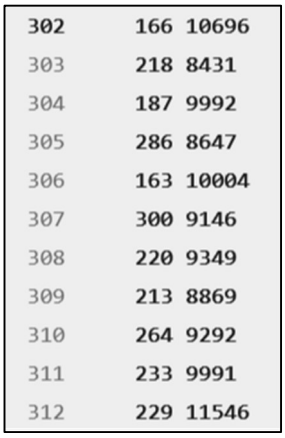


Figure 10 : Partie entrepôts

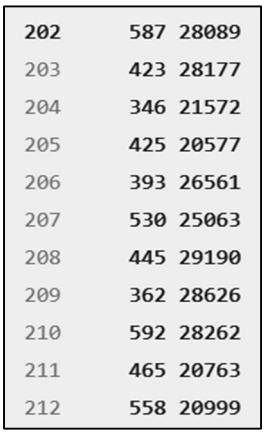


Figure 11 : Partie usines

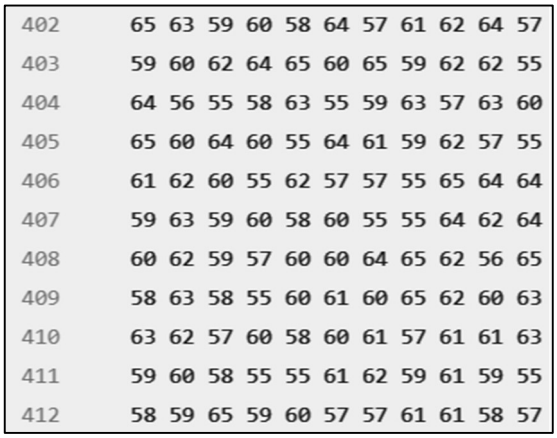


Figure 12 : Partie coût d'acheminement Entrepôts-Clients

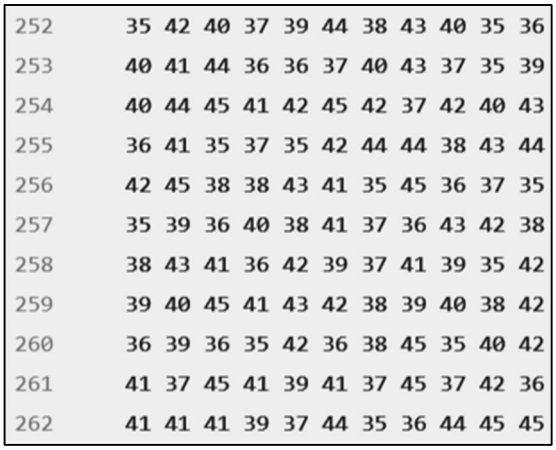


Figure 13 : Partie coût d'acheminement Usines-Entrepôts

4.3 Résultats des tests

Dans cette partie, nous allons exposer les résultats obtenus par notre algorithme :

4.3.1 Paramètres et détails de l'implémentation

L'algorithme PTS a été implémenté en JAVA, en utilisant un compilateur javaSE-17 (jdk-21). Nous avons choisi ce langage de programmation pour ses fonctions prédéfinies et parce qu'il nous a permis de nous concentrer sur la complexité algorithmique de notre travail.

Notre algorithme a été exécuté sur un ordinateur doté d'un processeur 13ème Génération Intel(R) Core i9 cadencé à 2.20 GHz, disposant de 32 GO de RAM.

En effet, le paramètre de randomisation $\boldsymbol{P}$ joue un rôle important dans l'algorithme proposé. Dans notre expérimentation, nous avons fixé le $\boldsymbol{P}$ de deux manières : (1) $\boldsymbol{P} = \mathbf{5}$, et (2) $\boldsymbol{P} = |\boldsymbol{N}(\boldsymbol{S})|$.

En conséquence, nous avons obtenu deux variantes de notre algorithme :

- PTSv1 : le PTS avec $\boldsymbol{P} = \mathbf{5}$.
- PTSv2 : le PTS avec $\boldsymbol{P} = |\boldsymbol{N}(\boldsymbol{S})|$.

Pour les deux variantes, le nombre d'itération est fixée à 1500 itération, et la liste tabou est calculer comme suit :

$$Taille\ de\ la\ liste\ tabou = 20 + |Solution.Entrepôts.Ouverts| + |Solution.Usines.Ouvertes|$$

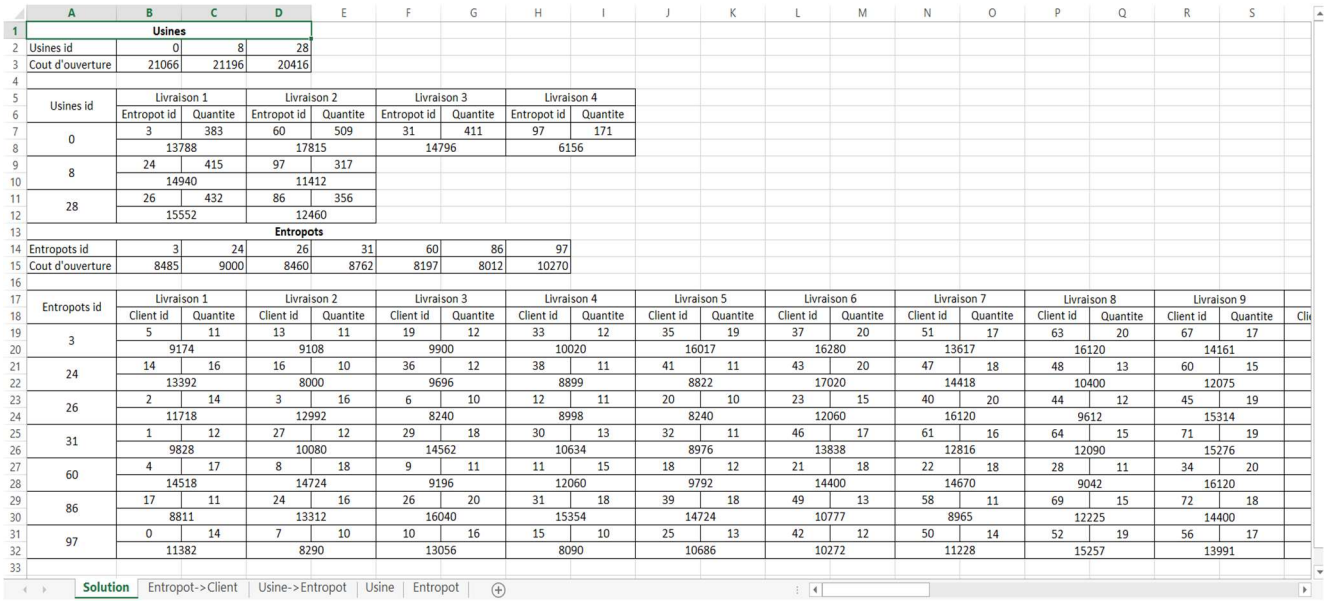
4.3.2 Structure de la solution

La solution générée inclut les Ids des locaux ouverts ainsi que les expéditions effectuées par chaque local ouvert à ses clients. Chaque livraison est composée de l'identifiant du client et de la quantité envoyée. Le tableau suivant présente la structure de la solution générée.

Niveau 1	Nbr d'usine ouverte	1	2	3	4	5	6	……	n-1	n
	Id de l'usine	0	9	11	22	25	31		44	45
	Id de l'usine ouverte	Livraison 1		Livraison 2		Livraison 3		……	Livraison n	
		Id de l'entrepôt	Quantité	Id de l'entrepôt	Quantité	Id de l'entrepôt	Quantité		Id de l'entrepôt	Quantité
	0 9 ⋮ 45	60 21 ⋮ 99	80 118 ⋮ 141	92 10 ⋮ 91	109 117 ⋮ 92	/ 65 ⋮ /	/ 1 ⋮ /	…… …… ……	65 98 ⋮ 78	109 44 ⋮ 65
Niveau 2	Nbr d'entrepôt ouvert	1	2	3	4	5	6	……	n-1	n
	Id de l'entrepôt	3	5	10	14	21	37		98	99
	Id de l'entrepôt ouvert	Livraison 1		Livraison 2		Livraison 3		……	Livraison n	
		Id du client	Quantité	Id du client	Quantité	Id du client	Quantité		Id du client	Quantité
	3 5 ⋮ 99	13 5 ⋮ 144	10 10 ⋮ 14	19 21 ⋮ 148	14 16 ⋮ 15	35 22 ⋮ 159	16 20 ⋮ 12	…… …… ……	66 85 ⋮ 199	5 4 ⋮ 17

Tableau 2 : Structure d'une solution générée

Une solution complète générée par l'algorithme PTS, ainsi que les coûts associés, sont illustrés dans la figure suivante.



Usines			
Usines id	0	8	28
Cout d'ouverture	21066	21196	20416

Usines id	Livraison 1		Livraison 2		Livraison 3		Livraison 4	
	Entropot id	Quantite	Entropot id	Quantite	Entropot id	Quantite	Entropot id	Quantite
0	3	383	60	509	31	411	97	171
	13788		17815		14796		6156	
8	24	415	97	317				
	14940		11412					
28	26	432	86	356				
	15552		12460					

Entropots						
Entropots id	3	24	26	31	60	86	97
Cout d'ouverture	8485	9000	8460	8762	8197	8012	10270

Entropots id	Livraison 1		Livraison 2		Livraison 3		Livraison 4		Livraison 5		Livraison 6		Livraison 7		Livraison 8		Livraison 9	
	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite	Client id	Quantite
3	5	11	13	11	19	12	33	12	35	19	37	20	51	17	63	20	67	17
	9174		9108		9900		10020		16017		16280		13617		16120		14161	
24	14	16	16	10	36	12	38	11	41	11	43	20	47	18	48	13	60	15
	13392		8000		9696		8899		8822		17020		14418		10400		12075	
26	2	14	3	16	6	10	12	11	20	10	23	15	40	20	44	12	45	19
	11718		12992		8240		8998		8240		12060		16120		9612		15314	
31	1	12	27	12	29	18	30	13	32	11	46	17	61	16	64	15	71	19
	9828		10080		14562		10634		8976		13838		12816		12090		15276	
60	4	17	8	18	9	11	11	15	18	12	21	18	22	18	28	11	34	20
	14518		14724		9196		12060		9792		14400		14670		9042		16120	
86	17	11	24	16	26	20	31	18	39	18	49	13	58	11	69	15	72	18
	8811		13312		16040		15354		14724		10777		8965		12225		14400	
97	0	14	7	10	10	16	15	10	25	13	42	12	50	14	52	19	56	17
	11382		8290		13056		8090		10686		10272		11228		15257		13991	

Figure 14 : Exemple de vérification manuelle d'une solution générée

4.3.3 Résultats obtenus et comparaison avec la littérature

Dans cette partie, nous allons présenter les résultats obtenus en exécutant 10 fois chacune des deux variantes de l'algorithme PTS sur chaque instance. Le premier tableau représente les résultats obtenus pour les instances de 50 usines, et le deuxième représente ceux de 100 usines. Voici quelques notations qui seront utilisées par la suite :

LowerB : il s'agit de la borne inférieure de l'instance.

GAP : un ratio calculé comme suit : $GAP = \frac{Cout\ de\ la\ solution - Borne\ inférieure}{Borne\ inférieure} \times 100$

AVG : représente la moyenne des GAP d'une instance.

BST : c'est le meilleur GAP obtenue lors de l'exécution de 10 fois sur la même instance.

TIME : c'est le temps moyen qu'il faut pour exécuter l'algorithme sur une instance.

Instances		LowerB	GA[47]		CS+CPLEX[48]			HEA / FA[59]			BRKGA + LS[50]			GRASPH[49]			CS-ALNS-LB[7]			SA[54]			PTS v1			PTS v2		
Classe	PSC		AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME
1	1	721209.6	0.13	264.78	0.13	0.22	48.90	0.13	0.24	431.34	0.14	0.14	19.32	0.13	0.13	4.50	0.13	0.13	15.41	0.29	0.44	40.64	4.19	5.49	120.45	1.58	3.28	117.34
	2	730451.6	0.40	257.17	0.23	0.31	76.86	0.31	0.36	510.74	0.24	0.24	115.93	0.24	0.24	21.34	0.23	0.23	21.02	0.44	0.62	36.26	2.91	5.70	114.02	2.94	4.59	122.13
	3	731885.3	0.24	263.35	0.21	0.29	51.94	0.22	0.24	327.60	0.22	0.22	39.29	0.22	0.22	30.46	0.21	0.21	55.22	0.49	0.68	41.24	5.80	6.33	122.82	2.46	2.60	116.51
	4	721515.0	0.81	242.93	1.19	1.41	96.62	0.54	0.59	472.15	0.50	0.51	116.29	0.50	0.50	29.08	0.50	0.50	18.98	0.69	0.76	34.05	5.81	7.26	114.77	1.92	2.32	89.82
	5	713633.8	0.82	251.79	0.81	0.88	56.14	0.86	0.97	480.59	0.81	0.81	59.10	0.81	0.81	160.01	0.81	0.82	64.69	1.59	1.72	36.49	4.65	8.58	117.37	3.65	4.66	110.50
2	1	479860.2	2.69	144.39	2.68	3.27	27.69	2.74	2.82	239.25	2.69	2.69	16.84	2.69	2.69	383.27	2.68	2.68	15.43	3.27	3.46	8.49	4.14	4.70	93.78	4.14	4.30	99.62
	2	483072.2	2.30	144.16	2.30	2.62	81.36	2.34	2.41	206.37	2.30	2.31	60.58	2.30	2.34	368.58	2.30	2.30	64.89	3.36	3.48	8.37	3.44	5.00	77.40	3.44	3.44	86.37
	3	486018.5	2.14	150.60	1.86	2.03	30.82	1.87	1.89	173.38	1.87	1.87	117.20	1.88	1.94	590.94	1.86	1.86	48.02	2.29	2.42	10.85	2.58	2.58	70.54	2.56	2.58	85.06
	4	482374.6	2.04	142.25	2.01	2.01	83.02	2.07	2.12	168.04	2.02	2.05	59.59	2.02	2.02	365.57	2.01	2.02	47.37	**2.65**	2.82	9.133	**2.28**	3.24	82.03	**2.28**	2.38	93.07
	5	474803.3	3.14	126.08	3.12	3.39	36.98	3.12	3.12	176.79	3.12	3.12	10.84	3.12	3.12	590.87	3.12	3.12	33.33	3.56	3.70	9.62	4.28	4.28	71.89	4.28	4.29	103.03
3	1	2608800	**3.07**	125.90	**3.07**	3.07	19.89	**3.14**	3.30	121.36	**3.11**	3.11	126.94	**3.22**	3.30	596.03	**3.07**	3.10	104.13	**3.72**	3.77	5.32	**3.07**	3.07	81.41	**3.07**	3.07	90.26
	2	2616252	**3.12**	130.22	3.10	3.10	73.09	**3.30**	3.36	120.92	**3.17**	3.17	40.26	**3.37**	3.39	594.73	**3.13**	3.20	94.18	**3.86**	3.93	5.23	**3.11**	3.11	83.75	**3.11**	3.11	94.11
	3	2598277	**3.11**	123.56	**3.09**	3.10	52.98	**3.30**	3.39	167.70	**3.10**	3.10	72.94	**3.23**	3.32	591.33	**3.09**	3.14	77.21	**4.04**	4.07	5.237	**3.11**	3.11	84.31	**3.09**	3.09	93.19
	4	2612534	**3.07**	107.73	3.05	3.05	45.21	**3.18**	3.22	154.29	**3.10**	3.10	8.25	**3.18**	3.29	593.68	3.05	3.10	80.60	**3.59**	3.66	5.22	**3.06**	3.06	81.46	**3.06**	3.06	91.99
	5	2568856	**3.01**	110.36	**3.01**	3.01	47.45	**3.17**	3.19	163.91	**3.13**	3.13	40.23	**3.14**	3.22	593.27	**3.01**	3.03	76.79	**3.65**	3.68	5.26	**3.01**	3.01	81.89	**3.01**	3.01	90.03
4	1	525294.1	3.14	138.25	3.14	3.60	89.47	3.36	3.54	224.71	3.14	3.22	58.44	3.29	3.29	591.54	3.14	3.14	39.42	**5.02**	5.27	18.50	**4.17**	4.42	86.16	**4.04**	4.13	96.01
	2	526911.7	2.33	139.83	2.43	2.71	102.38	2.74	2.92	206.66	2.43	2.51	80.98	2.65	2.80	592.19	2.43	2.45	57.23	**4.24**	4.35	16.82	**2.89**	3.10	72.33	**2.92**	3.24	91.20
	3	532592.3	2.66	144.88	2.41	2.44	118.38	2.59	2.62	256.30	2.33	2.42	110.29	2.45	2.92	591.35	2.30	2.44	67.49	**3.66**	3.94	21.12	**3.10**	3.23	73.55	**3.06**	3.11	90.69
	4	529372.0	2.53	127.30	2.35	2.66	133.69	2.63	2.81	271.29	2.44	2.51	74.85	2.36	2.50	591.31	2.35	2.36	55.50	**3.73**	3.92	18.41	**2.94**	3.28	70.56	**2.94**	2.98	88.85
	5	521470.1	3.13	120.27	3.15	3.53	115.67	3.18	3.18	188.88	3.14	3.16	39.57	3.15	3.23	388.72	3.12	3.12	46.46	3.77	4.21	18.99	3.97	4.42	83.52	3.88	4.16	94.89
5	1	2743547	1.20	164.42	1.19	1.19	157.20	1.16	1.20	206.49	1.18	1.22	120.41	1.24	1.31	591.33	1.16	1.18	64.60	**1.56**	1.60	21.87	**1.34**	1.36	72.87	**1.34**	1.36	96.06
	2	2752021	1.07	156.71	1.08	1.11	89.13	1.11	1.16	269.19	1.07	1.09	35.71	1.15	1.17	591.31	1.07	1.07	68.15	**1.45**	1.53	20.49	**1.23**	1.24	87.57	**1.23**	1.25	107.66
	3	2737769	1.10	191.60	1.09	1.10	126.33	1.22	1.28	210.10	1.13	1.15	130.43	1.29	1.30	591.78	1.09	1.13	70.37	**1.47**	1.55	25.37	**1.33**	1.35	95.03	**1.32**	1.34	111.29
	4	2748216	1.07	136.87	1.05	1.12	149.59	1.10	1.13	210.73	1.10	1.10	151.61	1.06	1.07	591.67	1.05	1.07	67.09	**1.37**	1.39	20.55	**1.28**	1.29	74.13	**1.27**	1.27	99.12
	5	2702350	1.25	145.07	1.24	1.24	54.96	1.31	1.34	188.84	1.26	1.27	74.20	1.29	1.34	592.50	1.23	1.25	67.90	**1.62**	1.70	21.74	**1.43**	1.44	85.28	**1.43**	1.43	104.18
Moyennes			1.98	162.01	1.95	2.09	78.63	2.02	2.09	245.90	1.94	1.96	71.20	1.99	2.05	449.09	1.92	1.94	56.85	2.61	2.74	18.61	3.16	3.74	87.95	2.72	2.96	98.51

Tableau 3 : Résultats obtenus pour le premier ensemble d'instances (50 usines, 100 entrepôts et 200 clients)

Instances		LowerB	GA[47]		CS+CPLEX[48]			HEA / FA[59]			BRKGA + LS[50]			GRASPH[49]			CS-ALNS-LB[7]			SA[54]			PTS v1			PTS v2		
Classe	PSC		AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME	BST	AVG	TIME
1	1	1475952	0.55	1268.12	0.10	0.30	384.79	0.29	0.33	1390.57	0.10	0.11	265.79	0.10	0.10	381.21	0.09	0.11	339.69	0.30	0.34	400.20	5.93	7.06	1028.09	2.56	3.34	895.56
	2	1462736	1.01	1250.09	0.34	0.70	716.06	0.27	0.43	1569.57	0.12	0.13	547.62	0.12	0.12	130.86	0.12	0.20	231.41	0.89	0.95	397.38	5.90	9.42	1121.54	2.53	3.03	884.33
	3	1492163	0.34	1367.04	0.54	1.00	654.10	0.23	0.48	1535.84	0.15	0.17	192.48	0.15	0.15	281.03	0.16	0.20	266.18	0.39	0.68	456.65	5.15	8.53	1185.97	3.44	5.10	1043.12
	4	1459076	0.49	1285.78	0.24	0.49	740.44	0.25	0.28	1163.24	0.22	0.23	284.96	0.22	0.28	484.13	0.22	0.24	240.00	0.45	0.53	428.89	8.29	9.82	1121.37	2.91	3.87	882.72
	5	1490742	0.67	1303.93	0.11	0.33	850.42	0.17	0.24	1555.82	0.12	0.12	515.29	0.12	0.12	82.79	0.11	0.12	192.28	0.27	0.31	413.74	7.60	9.13	1144.73	2.56	3.94	958.66
2	1	970908.5	0.89	675.48	0.26	0.52	989.39	0.63	0.70	979.27	0.27	0.28	398.44	0.27	0.36	584.20	0.26	0.30	310.07	0.58	0.64	75.51	2.58	4.16	769.10	1.89	3.41	770.73
	2	965908.5	0.74	662.96	0.28	0.46	668.55	0.47	0.56	941.41	0.29	0.29	458.38	0.28	0.34	559.24	0.28	0.330	257.51	0.71	0.80	75.16	4.45	5.67	819.12	2.57	3.31	743.57
	3	975499.7	1.42	650.19	0.14	0.25	992.58	0.20	0.25	955.61	0.14	0.15	89.55	0.14	0.14	487.39	0.14	0.17	236.91	0.39	0.46	73.61	3.64	4.87	777.89	2.88	3.67	823.08
	4	973019.1	0.56	657.63	0.28	0.40	688.28	0.56	0.59	1123.11	0.28	0.29	594.11	0.35	0.41	592.81	0.29	0.32	326.07	0.97	1.04	76.90	3.77	4.75	764.57	3.07	3.13	721.34
	5	941567.0	1.12	646.23	0.60	0.65	858.06	0.80	0.96	1201.59	0.61	0.61	506.97	0.86	1.06	592.22	0.60	0.69	283.63	1.48	1.55	76.96	2.89	4.20	680.51	2.88	2.89	647.81
3	1	5213566	1.63	617.24	1.62	1.63	1113.37	**1.91**	2.03	886.87	1.64	1.65	704.88	**1.79**	1.92	598.74	1.62	1.66	600.74	**2.07**	2.10	47.11	**1.65**	1.66	538.41	**1.65**	1.65	585.98
	2	5191321	1.67	601.51	1.65	1.65	1312.48	**1.96**	1.99	1264.97	1.68	1.71	234.85	**1.84**	1.94	596.33	1.67	1.71	503.07	**2.18**	2.19	46.59	**1.70**	1.70	550.66	**1.70**	1.70	594.41
	3	5145991	1.58	597.43	1.57	1.58	958.88	**1.78**	1.84	1236.61	**1.63**	1.63	90.64	**1.77**	1.84	594.41	1.58	1.61	491.88	**1.89**	1.90	46.37	**1.61**	1.61	532.55	**1.61**	1.61	574.70
	4	5225601	1.74	622.04	1.72	1.73	1033.46	**1.98**	2.01	1003.95	1.74	1.75	313.32	**2.01**	2.09	600.38	1.72	1.76	498.75	**2.15**	2.16	47.17	**1.76**	1.76	551.03	**1.76**	1.76	589.64
	5	5163182	**1.72**	629.84	1.67	1.69	1073.08	**1.99**	2.08	976.64	**1.72**	1.75	409.97	**2.02**	2.03	594.38	**1.73**	1.75	518.88	**2.15**	2.18	47.45	**1.72**	1.73	543.33	**1.72**	1.73	585.76
4	1	1052172	0.82	577.91	0.61	0.87	1040.75	0.91	1.04	910.67	0.64	0.69	437.63	0.73	0.74	594.13	0.65	0.78	313.52	1.85	2.21	200.26	2.21	2.83	574.15	2.20	2.29	649.47
	2	1043553	0.93	560.18	0.83	0.88	852.22	0.82	0.89	697.12	0.68	0.74	496.97	0.77	0.85	593.65	0.67	0.71	314.25	**1.88**	1.97	188.17	2.67	3.10	554.38	**1.65**	2.61	632.40
	3	1050683	1.88	584.40	0.62	0.81	677.12	1.00	1.29	909.24	0.73	1.04	566.19	1.20	1.77	592.49	0.78	1.12	299.23	**2.65**	2.84	195.95	**2.06**	3.03	592.12	**1.89**	2.44	632.96
	4	1044571	0.96	592.63	0.74	0.94	1099.22	1.36	1.62	826.32	0.81	0.88	573.03	0.98	1.01	594.38	0.80	0.90	303.95	**2.03**	2.20	205.05	2.66	2.91	539.67	**1.42**	2.57	601.94
	5	1053869	0.64	607.94	0.56	0.89	543.26	0.94	1.06	873.75	0.58	0.61	494.88	0.56	0.65	594.12	0.52	0.63	295.09	1.94	2.04	197.22	3.51	3.69	607.64	2.07	2.31	596.99
5	1	5486098	0.48	706.40	0.38	0.40	801.30	0.61	0.62	1002.84	0.39	0.41	358.97	0.43	0.54	593.59	0.38	0.40	326.89	0.63	0.67	286.15	0.70	0.88	657.83	0.71	0.88	758.65
	2	5461680	0.47	683.08	0.39	0.44	849.41	0.41	0.44	1054.17	0.41	0.43	500.54	0.40	0.42	593.52	0.38	0.41	285.26	0.60	0.63	286.64	0.81	0.81	652.87	0.81	0.81	695.15
	3	5425391	0.62	672.63	0.49	0.53	770.91	**0.64**	0.73	1421.97	0.42	0.45	555.96	0.59	0.59	594.61	0.39	0.48	259.29	**0.81**	0.84	317.31	**0.63**	0.92	666.20	**0.64**	0.86	765.37
	4	5494811	0.52	689.38	0.43	0.47	657.66	0.58	0.58	1123.96	0.43	0.44	363.07	0.50	0.51	593.16	0.41	0.43	322.00	0.69	0.70	291.60	0.78	0.86	656.05	0.78	0.82	710.40
	5	5442621	0.47	670.38	0.39	0.45	1323.85	0.43	0.47	1205.01	0.39	0.40	502.99	0.46	0.48	592.62	0.38	0.40	262.75	0.60	0.63	457.41	0.69	0.95	655.83	0.71	0.94	703.73
Moyennes			0.95	767.21	0.66	0.80	865.98	0.84	0.94	1112.40	0.64	0.67	418.29	0.74	0.81	523.85	0.63	0.69	331.17	1.22	1.30	213.41	3.01	3.84	731.42	1.94	2.42	721.93

Tableau 4 : Résultats obtenus pour le deuxième ensemble d'instances (100 usines, 200 entrepôts et 400 clients)

À présent, nous allons discuter les résultats obtenus par les variantes de PTS, puis nous comparerons ensuite ces résultats à ceux de la littérature :

a) Discussion des résultats

Lors de l'exécution de l'algorithme PTSv1, dans les deux catégories d'instances, nous avons constaté que les GAPs obtenus dans les classes 1 et 2 sont plus ou moins élevés. Pour les autres classes, elle a donné de bons GAPs. Notamment pour les instances de la classe 3, pour laquelle nous avons réussi à atteindre des bons GAPs.

Alors que l'algorithme PTSv2, nous a donné pour la première catégorie d'instances, de meilleures GAPs pour la classe 1 et des résultats similaires à ceux de la PTSv1 pour les autres classes. Pour la deuxième catégorie d'instances, cette dernière a donné de meilleurs GAPs pour les classes 1 et 2, et a donné des résultats relativement similaires à ceux de la PTSv1 pour les autres classes.

Nous remarquons alors que les deux méthodes se différencient au niveau des GAPs de la première et de la deuxième classe, pour lesquelles la PTSv2 a donnée de meilleures valeurs. Et de manière générale, on peut constater que le GAP moyen de la PTSv2 (2.72) est meilleur que celui de la PTSv1 (3.16) pour les instances de la première catégorie. De même pour ceux de la deuxième catégorie, où celui de la PTSv2 (1.94) devance celui de la PTSv1 (3.01).

Quant au temps d'exécution, nous avons noté que la PTSv1 était un peu plus rapide par rapport à la PTSv2 pour les instances de la première catégorie. Mais pour les instances de la deuxième catégorie, c'est la PTSv2 qui est plus rapide de quelques secondes.

b) Comparaison avec la littérature

Pour la PTSv1, nous pouvons remarquer que dans la première catégorie d'instances, nous avons obtenu une moyenne de GAPs de 3.16%, ce qui veut dire qu'elle est proche de l'algorithme génétique de 1.18%, et du CS+CPLEX de 1.21%, et du HEA / FA de 1.14%, et du BRKGA + LS de 1.22%, et du GRASPH de 1.17%, et du CS-ALNS-LB de 1.24% et du SA de 0.55%. Et pour les instances de la deuxième catégorie, elle a obtenu une moyenne de GAPs de 3.01%, elle est donc proche de l'algorithme génétique de 2.06%, et du CS+CPLEX de 2.35%, et du HEA / FA de 2.17%, et du BRKGA + LS de 2.37%, et du GRASPH de 2.27%, et du CS-ALNS-LB de 2.38% et du SA de 1.79%.

Alors que pour la PTSv2, nous pouvons remarquer que dans la première catégorie d'instances, nous avons obtenu une moyenne de GAPs de 2.72%, ce qui veut dire qu'elle est proche de l'algorithme génétique de 0.74%, et du CS+CPLEX de 0.77%, et du HEA / FA de 0.70%, et du BRKGA + LS de 0.78%, et du GRASPH de 0.73%, et du CS-ALNS-LB de 0.80% et du SA de 0.11%. Et pour les instances de la deuxième catégorie, elle a obtenu une moyenne de GAPs de 1.94%, elle est donc proche de l'algorithme génétique de 0.99%, et du CS+CPLEX de 1.28%, et du HEA / FA de 1.10%, et du BRKGA + LS de 1.30%, et du GRASPH de 1.20%, et du CS-ALNS-LB de 1.31% et du SA de 0.72%.

Instances	Méthodes Dépassées	Méthodes avec les même GAPs	Avons-nous atteint les meilleurs résultats ?
PSC1-C3-50	HEA / FA, BRKGA + LS, GRASPH (3/6)	GA, CS+CPLEX, CS-ALNS-LB (3/6)	✓
PSC2-C3-50	GA, HEA / FA, BRKGA + LS, GRASPH, CS-ALNS-LB (5/6)	/	✗
PSC3-C3-50	GA, HEA / FA, BRKGA + LS, GRASPH (4/6)	CS+CPLEX, CS-ALNS-LB (2/6)	✓
PSC4-C3-50	GA, HEA / FA, BRKGA + LS, GRASPH (4/6)	/	✗
PSC5-C3-50	HEA / FA, BRKGA + LS, GRASPH (3/6)	GA, CS+CPLEX, CS-ALNS-LB (3/6)	✓
PSC1-C3-100	HEA / FA, GRASPH (2/6)	/	✗
PSC2-C3-100	HEA / FA, GRASPH (2/6)	/	✗
PSC3-C3-100	HEA / FA, BRKGA + LS, GRASPH (3/6)	/	✗
PSC4-C3-100	HEA / FA, GRASPH (2/6)	/	✗
PSC5-C3-100	HEA / FA, GRASPH, CS-ALNS-LB (3/6)	GA, BRKGA + LS (2/6)	✗
PSC3-C5-100	HEA / FA (1/6)	/	✗

Tableau 5 : Comparaison des meilleurs résultats obtenus avec ceux de la littérature

Dans le tableau 5 nous remarquons que dans les instances PSC1-C3-50, PSC3-C3-50 et PSC5-C3-50 nous avons atteint les meilleurs résultats de la littérature en obtenant des résultats meilleurs que quelques méthodes de la littérature. Par exemple, pour l'instance PSC3-C3-50, on a trouvé des résultats meilleurs que les algorithmes : GA, HEA / FA, BRKGA + LS et GRASPH. et nous avons obtenu les même meilleurs résultats trouvé par les algorithmes CS + CPLEX et CS-ALNS-LB.
Et malgré qu'on ait pas trouvé les meilleurs résultats de la littérature dans les instances PSC2-C3-50, PSC4-C3-50, PSC1-C3-100, PSC2-C3-100, PSC3-C3-100, PSC4-C3-100, PSC5-C3-100 et PSC3-C5-100, on a trouvé des résultats meilleurs que certaines méthodes. Par exemple, pour l'instance PSC5-C3-100, on a réussi à obtenir de meilleurs résultats que les algorithmes : HEA / FA, GRASPH et CS-ALNS-LB. et nous avons obtenu les même meilleurs résultats trouvé par les algorithmes GA et BRKGA + LS.

Pour le temps d'exécution, nous pouvons remarquer que dans la première catégorie d'instances, la PTSv1 avait une moyenne de 87.95 secondes, une moyenne meilleure que celle des algorithmes GA, HEA / FA et GRASPH, alors que pour la deuxième catégorie d'instances, elle a un temps d'exécutions meilleurs que GA, CS+CPLEX et HEA / FA, avec une moyenne de 731.42 secondes. Alors que la PTSv2 avait une moyenne de 98.51 secondes pour les instances de la première catégorie, une meilleure moyenne que celle des algorithmes GA, HEA / FA et GRASPH, alors que pour la deuxième catégorie d'instances, elle a un temps d'exécutions meilleurs que GA, CS+CPLEX et HEA / FA, avec une moyenne de 721.93 secondes.

4.4 Conclusion

Dans ce chapitre, nous avons expliqué comment les instances de notre problème ont été générées. Puis nous avons présenté quelques exemples d'instances. Après, nous avons donné les paramètres et détails de notre implémentation que nous avons utilisée. Suite à quoi nous avons donné la structure d'une solution. Enfin, nous avons présenté les résultats obtenus, en les comparants à ce qui s'est fait de plus important dans la littérature.

Conclusion générale

Dans ce mémoire, nous avons proposé un algorithme pour résoudre le problème de localisation à deux niveaux avec contrainte de capacité (TSCFLP). L'objectif principal était de trouver un ensemble de locaux capable de satisfaire les demandes de tous les clients, avec un coût d'installation et de transport des produits réduit.

Dans le premier chapitre, nous avons défini les problèmes de localisation en présentant les plus importants, tels que le problème de localisation avec contrainte de capacité, le problème du p-médiane, le problème de couverture maximale. Puis, nous avons fourni une description du problème de localisation à deux niveaux avec contrainte de capacité, avant de présenter quelques applications des problèmes de localisation les plus courantes dans la vie réelle et industriel.

Puis, dans le deuxième chapitre, nous avons donné une définition des problèmes d'optimisation et vu les heuristiques et les métaheuristiques les plus utilisées pour les résoudre. Ensuite, nous avons défini les méthodes exactes en présentant la méthode du Branch and Bound. Enfin, nous avons présenté quelques techniques d'hybridation.

Dans le troisième chapitre, nous avons définit la variante du problème de localisation à deux niveaux avec contrainte de capacité (TSCFLP) traité dans ce mémoire. Puis nous avons revu les travaux de la littérature qui ont abordés cette variante. Après, nous avons détaillé l'algorithme de recherche tabou probabiliste qu'on a proposé pour résoudre le TSCFLP pour lequel nous avons proposé deux versions.

Dans le quatrième chapitre, nous avons évalué l'efficacité de notre algorithme à travers une comparaison avec la littérature. En effet, notre algorithme a donné des résultats prometteurs par rapport à la littérature. Cette comparaison a démontré la robustesse et la compétitivité de notre approche, ouvrant ainsi des perspectives pour de futures travaux.

Enfin, nous envisageons de :

- Améliorer notre algorithme en utilisant des techniques d'allocations.
- Résoudre ce problème en utilisant d'autres métaheuristiques tel que : la méthode de recherche à voisinage variable.
- Proposer un algorithme hybride entre les deux versions de PTS proposées et un autre algorithme.
- Renforcer notre algorithme afin d'obtenir de meilleurs résultats pour les instances de classe 1 et 2.

Bibliographie

[1] A. Klose et A. Drexl, «Facility location models for distribution system design,» European Journal of Operational Research , vol. 162, p. 4–29, 2005.

[2] C. Ortiz-Astorquiza, I. Contrerasn et G. Laporte, «Multi-level Facility Location Problems Volume» European Journal of Operational Research, vol. 267, n° %13, pp. 791-805, 16 June 2018.

[3] S. Basu, M. Sharma et G. P. Sarathi, «, Metaheuristic applications on discrete facility location problems: a survey» Operational Research Society of India, 2014.

[4] J. Current, H. Min and D. Schilling, "Multiobjective analysis of facility location decisions," European Journal of Operational Research, vol. 49, p. 295–307, 1990.

[5] C. S. Revelle et G. Laporte, «The plant location problem: New models and research prospects» Operations Research, vol. 44 , n° %16, p. 864–874, 1996.

[6] E. Fernández et J. Puerto, «Multiobjective solution of the uncapacitated plant location problem» European Journal of Operational Research, vol. 145, p. 509–529, 2003.

[7] P. H. González, I. M. Gabriel Souto, G. R. Mauri and G. M. Ribeiro, "A hybrid matheuristic for the Two-Stage Capacitated Facility Location problem" Expert Systems With Applications, vol. 185, 2021.

[8] F. Vasko, D. Newhart, K. Stott and F. Wolf, "A large-scale application of the partial coverage uncapacitated facility," Journal of the Operational Research Society, vol. 54, p. 11–20, 2003.

[9] B. Boffey, D. Yates et R. D. Galvão, « An algorithm to locate perinatal facilities in the municipality of Rio de Janeiro. Journal of» Journal of the Operational Research Society, vol. 54, pp. 21-31, 2003.

[10] J. M. Mulvey et H. P. Crowder, «Cluster analysis: An application of Lagrangean relaxation. Management Science,» vol. 25, n° %14, p. 329–340, 1979.

[11] G. Cornuejols, M. L. Fisher and G. L. Nemhauser, "Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms," Management Science, vol. 23, p. 789–810, 1977.

[12] J. Current et C. Weber, «Application of facility location modeling constructs to vendor selection problems,» European Journal of Operational Research, vol. 76, n° %13, p. 387– 392, 1994.

[13] P. Hansen, E. d. L. P. Filho et C. C. Ribeiro, «Location and sizing of offshore platforms for oil exploration.,» European Journal of Operational Research, vol. 58, p. 202–214, 1992.

[14] P. Hansen, E. d. L. P. Filho et C. C. Ribeiro, «Modelling location and sizing of offshore platforms,» European Journal of Operational Research, vol. 72, p. 602–606, 1994.

[15] M. L. Fisher et D. S. Hochbaum, «Database location in computer networks,» Journal of the ACM , vol. 7, p. 718–735, 1980.

[16] T. Boffey, «Location problems arising in computer networks.,» The Journal of the Operational Research Society, vol. 40 , n° %14, p. 347–354, 1989.

[17] B. Gavish, «Topological design of telecommunication networks—local access design methods,» Annals of Operations Research, vol. 33, p. 17–71, 1991.

[18] A. Mirzaian, «Lagrangian relaxation for the star-star concentrator location problem: Approximation algorithm and bounds,» Networks , vol. 15, p. 1–20, 1985.

[19] H. Pirkul, «Efficient algorithms for the capacitated concentrator location problem,» Computers & Operations Research, vol. 14, p. 197– 208, 1987.

[20] A. Caprara, M. Fischetti et D. Maio, «Exact and approximate algorithms for the index selection problem in physical database design,» IEEE Transactions on Knowledge and Data Engineering, vol. 7, n° %16, p. 955–967, 1995.

[21] O. Guemri, Proposition de solutions pour l'optimisation, Département d'Informatique Laboratoire d'Informatique d'Oran (LIO) éd., Thèse du Doctorat en LMD , 2017.

[22] A. H. Land et A. G. Doig, «An automatic method of solving discrete programming problems,» In: Econometrica: Journal of the Econometric Society , p. 497– 520, 1960.

[23] J. Clausen, Branch and Bound Algorithms-Principles and Examples, Copenhagen, Denmark: Department of Computer Science, University of Copenhagen, 1999, p. 1–30.

[24] E. A. Feigenbaum et J. Feldman, Computers and thought, New York,NY, USA: McGrawHill, 1963.

[25] M. G. C. Resende et C. C. Ribeiro, «Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications,» chez Handbook of Metaheuristics, 2010, pp. 283-319.

[26] H. Osman et G. Laporte, « Metaheuristics: a bibliography,» Annals of Operations Research, vol. 63, pp. 513-623, 1996.

[27] S. Kirkpatrick, C. Gelatt et M. Vecchi, «Optimization by Simulated Annealing,» Science, new series, p. 671 – 680 , 1983.

[28] F. Glover, «Future paths for integer programming and links to artificial intelligence,» Computers and Operations Research, vol. 13, pp. 533-549, 1986.

[29] T. Feo et M. Resende, «A probabilistic heuristic for a computationally difficult set covering problem,» Operations Research Letters, vol. 8, p. 67–71, 1989.

[30] N Mladenovi´c et P. Hansen, «Variable neighborhood search,» Computers & Operations Research, vol. 24, p. 1097–1100, 1997.

[31] N. Mladenovi´c, P. Hansen et J. M. Pérez, « Variable neighbourhood search: Methods and applications,» Annals of Operations Research, vol. 175, p. 367–407, 2010

[32] R. Kammarti, Evolutionary approaches for the resolution of static 1-PDPTW and dynamic, Doctoral thesis, Ecole Centrale de Lille, France, 2006.

[33] Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press: Ann Arbor, 1975.

[34] N. M. Razali et J. Geraghty, A genetic algorithm performance with different selection strategies, 2 éd., vol. 2, Proceedings of the World Congress on Engineering , 2011.

[35] J. Kennedy et R. Eberhart, «Particle swarm optimization,» IEEE International Conference on Neural Network, p. 1942–1948, 1995.

[36] M. Dorigo, G. D. Caro et L. Gambardella, «Ant algorithms for discrete optimization,» Artificial Life, vol. 5, n° %12, p. 137–172, 1999.

[37] G. Talbi, «Combining metaheuristics with mathematical programming, constraint programming and machine learning,» Annals of Operations Research, vol. 240, n° %11, p. 171–215, 2016.

[38] A. E. Samrout, hybridization of multicriteria metaheuristic optimization methods, universite de technologie de troyes, 2018.

[39] J. Puchinger et G. R. Raidl, «Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification,» Springer, p. 41, 2005.

[40] E.-G. Talbi, «Metaheuristics: from design to implementation,» John Wiley & Sons, vol. 74, 2009.

[41] E. Alba, G. Luque et S. Nesmachnow, «Parallel metaheuristics: recent advances and new trends,» International Transactions in Operational Research, vol. 20, n° %11, p. 1–48, 2013.

[42] W. Bożejko, J. Pempera et C. Smutnicki, «Parallel tabu search algorithm for the hybrid flow shop problem,» Computers & Industrial Engineering , vol. 65, n° %13 , p. 466–474, 2013.

[43] M. Hijaze et D. Corne, «An investigation of topologies and migration schemes for asynchronous distributed evolutionary algorithms,» Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on. IEEE. , p. 636– 641, 2009.

[44] Y.-L. Chang, K.-S. Chen, B. Huang et W.-Y. Chang, «A parallel simulated annealing approach to band selection for high-dimensional remote sensing images,» IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 4, n° %13, p. 579–590, 2011.

[45] T. O. Ting, X.-S. Yang, S. Cheng et K. Huang, «Hybrid metaheuristic algorithms: past, present, and future",» Recent Advances in Swarm Intelligence and Evolutionary Computation. Springer, p. 71–83, 2015.

[46] P. Shelokar, P. Siarry, V. Jayaraman et B. Kulkarni, «Particle swarm and ant colony algorithms hybridized for improved continuous optimization,» Applied mathematics and computation, vol. 188, n° %11 , p. 129–142, 2007.

[47] D. R. Fernandes, C. Rocha, D. Aloise, G. M. Ribeiro, E. M. Santos et A. Silva, «A simple and effective genetic algorithm for the two-stage capacitated facility location problem,» Computers & Industrial Engineering, vol. 75, p. 200–208, 2014.

[48] R. R. Louzada, G. R. Mauri et G. M. Ribeiro, «M´etodo heurístico híbrido para resoluç˜ao do problema de localizaçao ˜ de facilidades capacitadas em dois níveis,» Simposio ´ Brasileiro de Pesquisa Operacional - SBPO, p. 2460–2471, 2016.

[49] P. H. González, G. S. d. J. Augusto, G. R. Mauri, G. Ribeiro et L. G. Simonetti, «Grasp híbrido para resoluçao ˜ do problema de localizaç˜ ao de facilidades capacitadas em dois níveis.,» Simposio ´ Brasileiro de Pesquisa Operacional - SBPO, p. 1–11, 2019.

[50] Biajoli, F. L., Chaves, A. A., & Lorena, L. A. N. (2019). A biased random-key genetic algorithm for the two-stage capacitated facility location problem. Expert Systems with Applications, 115, 418-426.

[51] Kumar, B., & Kumar, D. (2013). A review on Artificial Bee Colony algorithm. International Journal of Engineering & Technology, 2(3), 175.

[52] Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. Advances in engineering software, 95, 51-67.

[53] Farahani, R. Z., Hekmatfar, M., Arabani, A. B., & Nikbakhsh, E. (2013). Hub location problems: A review of models, classification, solution techniques, and applications. Computers & industrial engineering, 64(4), 1096-1109.

[54] Tariq, A. (2023). Algorithms for the Two-Stage Capacitated Facility Location Problem (university center of abdalhafid boussouf-MILA).

[55] Yildiz, S. T., Ozcan, S., & Cevik, N. (2023). Variable neighborhood search-based algorithms for the parallel machine capacitated lot-sizing and scheduling problem. Journal of Engineering Research, 100145.

[56] Jackson, L. E., Rouskas, G. N., & Stallmann, M. F. (2007). The directional p-median problem: Definition, complexity, and algorithms. European Journal of Operational Research, 179(3), 1097-1108.

[57] Berman, O., & Krass, D. (2002). The generalized maximal covering location problem. Computers & Operations Research, 29(6), 563-581.*

[58] Bachelet, V., Hafidi, Z., Preux, P., & Talbi, E. G. (1998). Vers la coopération des métaheuristiques. Calculateurs parallèles, réseaux et systèmes répartis, 9(2), 211-223.

[59] Guo, P., Cheng, W., & Wang, Y. (2017). Hybrid evolutionary algorithm with extreme machine learning fitness function evaluation for two-stage capacitated facility location problems. Expert Systems with Applications, 71, 57-68.

[60] Guemri, O., Nduwayo, P., Todosijević, R., Hanafi, S., & Glover, F. (2019). Probabilistic tabu search for the cross-docking assignment problem. European Journal of Operational Research, 277(3), 875-885.