REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IA 23/M2/2019

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **IA**

L'algorithme d'optimisation par essaim d'abeilles pour la résolution du problème d'arbre dominant

Par :

DJERAD AMAR

Soutenu le 7 juillet 2019, devant le jury composé de :

KALFALI Toufik	MAA	Président
BERGHIDA Meryem	MCB	Rapporteur
GUERROUF Fayçal	MAA	Examinateur

Remerciement :

*Nous remercions avant tout, « **Allah** » le tout puissant qui nous a éclairé le chemin de la réussite et du savoir et nous a donné le courage et la volonté pour réaliser ce travail.*

*Nous remercions tout d'abord notre directrice de mémoire **Dr Meryem Berghida** qui a accepté de nous encadrer et pour ses conseils et se orientation et son aide.*

*Nous tenons à remercier également les membres du jury Mr **Kalfali Toufik** et Mr **Guerrouf Fayçal**, pour avoir accepté de juger ce mémoire.*

DEDICACE

Après cette réussite que fait la joie
A tous qui m'aime . Je dédie ce modeste travail avec vif plaisir à ceux qui sont les plus prochesà mon cœur, qui mon toujours aimer et qui ont fait de toute leurs force pour que je sois toujours heureux. mes très chères parents, ma mère que Dieu la protège et mon père qu Allah lui fasse miséricorde , qui sans eux je serai rien

À mes frères et soeurs ;

À toute ma grande famille ;

À mes amies.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

Au cours des dernières années, de nombreux problèmes d'optimisation combinatoire ont été rencontrés dans le domaine des réseaux de capteurs sans fil (WSN). Le problème de l'arbre dominant (DTP) est l'un des problèmes NP-Difficile dans les WSN.

Une solution au DTP propose une application pour fournir un squelette virtuel pour le routage dans les réseaux de capteurs. Ce problème consiste à trouver un arbre, disons DT, avec un poids minimum total des arêtes sur un graphe non orienté, pondéré et connexe, de sorte que chaque sommet du graphe soit en DT ou adjacent à un sommet en DT.

Il existe des procédures heuristiques efficaces pour ce problème qui atteignent de bonnes solutions en un temps de calcul très court pour les cas avec des centaines de nœuds. Néanmoins, du point de vue des modèles de programmation en nombres entier, seuls quelques modèles exponentiels disponibles peuvent être explorés dans des méthodes exactes pour prouver l'optimalité des instances DT.

Le problème DT comporte plusieurs applications dans la conception de réseau et le routage réseau. Par exemple, la multidiffusion implique la distribution des mêmes données à partir d'un serveur central à plusieurs nœuds du réseau. Dans ce contexte, nous pouvons considérer le poids d'arête comme la consommation d'énergie pour envoyer un message le long de cette arête. Ainsi, le problème devient choisir un ensemble d'arêtes (ou de liens de communication) de poids minimum pour que le serveur achemine les données, ce qui est exactement le problème de DT. Étant donné que tous les nœuds sont au plus à un bond de l'arbre, un message peut être transmis au nœud le plus proche dans l'arbre. Ensuite, le message peut être acheminé dans le DT jusqu'à ce qu'il atteigne sa destination. En outre, l'utilisation de DT comme squelette de routage peut aider à réduire la complexité des frais généraux du message.

De nos jours, il existe de nombreux algorithmes d'optimisation qui fonctionnent en utilisant des techniques de recherche à base de gradient et heuristiques dans des contextes

déterministes et stochastiques. Afin d'élargir l'applicabilité de l'approche d'optimisation à divers domaines problématiques, les principes naturels et physiques sont imités pour développer des algorithmes d'optimisation robustes.

Les algorithmes d'optimisation, le recuit simulé, l'optimisation des colonies de fourmis, l'optimisation par essaims d'abeilles, les algorithmes mémétiques, l'optimisation des essaims de particules sont quelques exemples de tels algorithmes.

Au cours de la dernière décennie, les algorithmes d'optimisation inspirés de la nature ont été largement utilisés dans différents domaines problématiques et ont réussi à trouver efficacement les solutions quasi optimales.

Nous nous intéressons dans ce travail à la résolution approchée du problème de l'arbre dominant par un algorithme d'optimisation par essaims d'abeilles BSO. Le problème ainsi défini est noté DTP. BSO est une metaheuristique récente qui s'inspire du comportement des abeilles pour la recherche de la nourriture.

L'objectif de ce mémoire est d'adapter l'algorithme d'optimisation par essaims d'abeilles pour résoudre le problème de l'arbre dominant DTP. Pour cela, nous proposons un encodage adéquat de la solution et nous décrivons les différentes étapes de l'approche proposée.

Ce mémoire est constitué de cinq chapitres. Les trois premiers sont consacrés aux études théoriques. Le quatrième chapitre décrit l'approche proposée et le cinquième présente les résultats d'expérimentations sur différentes instances prises de la littérature. Une brève description de ces chapitres est faite dans les paragraphes suivants :

Dans le premier chapitre, nous présentons des notions liées à l'optimisation combinatoire. Ensuite, nous introduisons la notion des métaheuristiques en présentant des généralités sur ces derniers.
Dans le chapitre 2 nous présentons en détails les étapes de l'algorithme d'optimisation par essaims d'abeilles.

Dans le chapitre 3, nous présentons quelques notions de bases sur les graphes. Ensuite, nous présentons le problème de l'arbre dominant et les problèmes connexes au problème DT.

Dans le chapitre 4, nous développons l'approche proposée pour résoudre le problème de l'arbre dominant (DTP), pour cela, nous présentons l'approche proposée, à savoir, l'algorithme d'optimisation par essaims d'abeilles. Le chapitre 5 est réservé à l'implémentation de la méthode proposée (environnement de travail, données de test, résultats et comparaison avec d'autres méthodes). Ce mémoire est clôturé par une conclusion générale et perspectives.

Chapitre 1

Introduction à l'optimisation

Sommaire

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire [2]. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données [3]. Etant donnée l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA).

Ces méthodes peuvent être classées sommairement en deux grandes catégories : les méthodes exactes (complètes) qui garantissent la complétude de la résolution et les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité.

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de séparation et évaluation progressive (SEP) ou les algorithmes avec retour arrière.

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante.

Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens. On peut citer les méthodes gloutonnes et l'amélioration itérative : par exemple, la méthode de Lin et Kernighan qui resta longtemps le champion des algorithmes pour le problème du voyageur de commerce [4]. Depuis une dizaine d'années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées métaheuristiques [2]. Une métaheuristique est constituée d'un ensemble de concepts fondamentaux (par exemple, la liste tabou et les mécanismes d'intensification et de diversification pour la métaheuristique tabou), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation1. Ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Les métaheuristiques sont représentées essentiellement par les méthodes de voisinage comme le recuit simulé et la recherche tabou, et les algorithmes évolutifs comme les algorithmes génétiques et les stratégies d'évolution. Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant [2]. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle.

1.1 L'optimisation combinatoire

L'optimisation combinatoire tient une place importante dans la recherche opérationnelle, en mathématiques discrètes et en informatique. En effet, les problèmes d'optimisation combinatoire représentent une catégorie de problèmes très difficiles à résoudre [5] et de nombreux problèmes pratiques peuvent être formulés sous la forme d'un problème d'optimisation [6]. De plus, l'étude de ces problèmes représente un gain non négligeable sur la qualité de leur résolution.

Un problème d'optimisation combinatoire, consiste dans un espace discret (i. c. énumérable) de solutions réalisables, à trouver la meilleure solution (ou un ensemble des meilleures solutions). La notion de « meilleure solution » est donnée par un critère de qualité via une fonction objectif. Formellement, un problème d'optimisation combinatoire peut être défini par : *L'ensemble discret des solutions réalisables du problème*, on parle alors d'**espace de recherche** , et $f : \Omega \rightarrow R$ **la fonction objectif** associée au critère de qualité. Le but est de trouver $s^* \in \Omega$ tel que :

$$\begin{aligned} s^* &= \arg\max\{f(s)\} \\ s^* &\in \Omega \end{aligned} \tag{1.1}$$

On parle d'un problème de **maximisation** quand la qualité est donnée par une fonction objectif à maximiser et d'un problème de **minimisation** quand la qualité est donnée par une fonction objectif à minimiser.

La valeur de **fitness** d'une solution de l'espace de recherche est la valeur de la fonction objectif pour cette solution. L'évaluation d'une solution correspond au calcul de sa valeur de fitness.

Dans ce contexte, nous définissons les termes d'**optimum global** et d'**optimum local** associés à un problème d'optimisation. Pour un problème de minimisation (resp. maximisation), un **optimum global** est une solution $s^* \in \Omega$ tel que :

$$\forall s \in \Omega \ f(s^*) \leq f(s) \tag{1.2}$$

$$(resp.\ f(s^*) \geq f(s)) \tag{1.3}$$

Si l'espace de recherche Ω est muni d'une relation de **voisinage** V. Un **optimum local** est alors défini comme une solution $s^* \in \Omega$ tel que :

$$\forall s \in V(s^*),\ f(s^*) < f(s) \tag{1.4}$$

$$(resp.\ f(s^*) > f(s)) \tag{1.5}$$

Les problèmes d'optimisation peuvent être rencontrés dans plusieurs domaines comme les sciences, l'ingénierie, la gestion ... etc. Par exemple, un commerçant doit livrer des articles chez ses clients partout en France et souhaite minimiser le coût lié à la livraison. Il dispose de plusieurs camions et doit affecter à chacun d'eux une liste de clients à livrer. Ceci peut être modélisé par un problème d'optimisation combinatoire où l'ensemble des solutions réalisables correspond à tous les chemins de livraisons possibles et où la distance parcourue par l'ensemble des camions doit être minimale. Ce problème d'optimisation, identifié comme *un problème de tournées de véhicules*, voit sa difficulté s'accroître à mesure que le nombre de clients augmente.

Une **instance** d'un problème est obtenue en donnant des valeurs explicites pour chacun des paramètres du problème instancié. Dans l'exemple précédent, une instance du problème peut correspondre à une liste de clients et, d'articles à livrer et un ensemble de véhicules disponibles le jour considéré. Le jour suivant, l'instance sera différente et la résolution du problème également.

Un algorithme est une séquence d'opérations élémentaires (affectation de variable, tests, etc.) qui, quand on le donne une instance d'un problème en entrée, il donne la solution de ce problème en sortie après l'exécution de l'opération finale.

Les deux paramètres les plus importants pour mesurer la qualité d'un algorithme sont *son temps d'exécution* et *l'espace de mémoire* qui est utilisée. Le premier paramètre est exprimé en termes de nombre d'instructions nécessaires pour exécuter l'algorithme. Le second paramètre correspond au nombre d'unités de mémoire utilisées par l'algorithme afin de résoudre un problème.

L'utilisation du nombre d'instructions comme unité de temps se justifie par le fait que le même programme utilise le même nombre d'instructions sur deux machines différentes, mais le temps peut varier, en fonction des vitesses respectives des machines. Nous considérons généralement qu'une instruction correspond à une opération élémentaire, par exemple une affectation, un test, une addition, une multiplication, etc. Ce que nous appelons la **complexité en temps** ou tout simplement la **complexité d'un algorithme** nous donne une indication sur le temps qu'il faudra pour résoudre un problème d'une taille donnée. La complexité est généralement définie en termes d'analyse au pire des cas.

La complexité d'un problème est équivalente à la complexité du meilleur algorithme résolvant ce problème. Un problème est traitable (ou facile) s'il existe un algorithme polynomial pour le résoudre.

Une grande partie des problèmes d'optimisation combinatoire font partie des problèmes NP-difficiles pour lesquels il n'existe pas d'algorithme de résolution efficace pour toutes les instances [7]. Il existe deux classes de méthodes pour résoudre ces problèmes

d'optimisation, les méthodes :

- **Les méthodes exactes** : elles garantissent la complétude de la résolution,
- **Les méthodes approchées** : elles perdent la complétude mais gagnent en efficacité.

Ces méthodes sont souvent testées au préalable sur des instances de la littérature. Ces instances permettent de comparer les méthodes et leur efficacité. On appelle **Best-Known Solution BKS**, la meilleure solution connue d'une instance. Cette solution peut être la meilleure parmi les solutions de l'espace de recherche, c'est donc *une solution optimale*, ou *la meilleure solution trouvée*.

Les méthodes *exactes* énumèrent implicitement toutes les solutions de l'espace de recherche en utilisant des mécanismes qui détectent des échecs (*calcul de bornes*) et des heuristiques spécifiques au problème qui orientent les choix.

Ces méthodes ont permis de trouver des solutions optimales. Mais ces méthodes s'avèrent, malgré les progrès réalisés, plutôt inefficaces à mesure que la taille du problème devient importante.

Les méthodes *approchées* constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, elles permettent d'obtenir des solutions de qualité intéressante en un temps de calcul réduit.

Les métaheuristiques font partie de ces méthodes approchées [8]. Les métaheuristiques sont composées de concepts fondamentaux qui permettent de les adapter et de les appliquer à une large classe de problème d'optimisation.

1.2 Les heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial au moins une solution réalisable rapide, pas obligatoirement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée. Les heuristiques peuvent être classées en deux catégories :

• **Méthodes constructives** qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.

• **Méthodes de fouilles locales** qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette

solution en explorant son voisinage.

1.3 Généralités Sur Les Métaheuristiques

Si certaines heuristiques sont spécifiques à un problème, d'autres ont pour vocation de pouvoir être adaptées à divers problèmes. On appelle parfois ces dernières des métaheuristiques ou encore heuristiques générales. En fait, il s'agit de principes algorithmiques permettant d'obtenir une solution en respectant certains principes de construction. Les principales sont les méhodes gloutonnes ; les méthodes de recherche locale (ou d'améliorations itératives) telle que la méthode tabou (ou ses améliorations comme Grasp) ou les méthodes de descentes (recuit simulé) ; les méthodes évolutives (algorithmes génétiques) ; et la simulation (objet ou continue).

Finalement, mentionnant que certaines métaheuristiques utilisent les concepts additionnels que sont la diversification et l'intensification.

La diversification (ou exploration, synonyme utilisé presque indifféremment dans la littérature) désigne les processus visant à explorer différentes zones dans l'espace de recherche du problème à optimiser.

L'intensification (ou exploitation) vise à parcourir une zone de l'espace de recherche pour trouver la meilleure solution. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux.

Les métaheuristiques progressent de façon itérative, en alternant des phases d'intensification, de diversification et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint. Les notions d'intensification et de diversification sont prépondérantes dans la conception des métaheuristiques, qui doivent atteindre un équilibre délicat entre ces deux dynamiques de recherches. Les deux notions ne sont donc pas contradictoires, mais complémentaires, et il existe de nombreuses stratégies les mêlant.

Les métaheuristiques présentent une classe des méthodes simples à mettre en œuvre, comme le cas du recuit simulé, d'autres sont bien adaptées à la résolution de certaines classes de problèmes...etc. Selon le « No Free Lunch Theorem[Ho.and.Pepyne,2002] »[9],il n'existe pas de métaheuristique qui soit meilleure que toutes le autres métaheuristiques pour tous les problèmes. Dans la pratique, il existera toujours des instances pour lesquelles une métaheuristique est meilleure qu'une autre. Quel que soit la métaheuristique choisie, elle présente des avantages et des inconvénients.

1.3.1 Classification

On peut faire la différence entre les métaheuristiques qui s'inspirent de phénomènes naturels et celles qui ne s'en inspirent pas. Par exemple, les algorithmes génétiques et les algorithmes de colonies de fourmis s'inspirent respectivement de la théorie de l'évolution et du comportement de fourmis à la recherche de nourriture. Par contre, la méthode Tabou ne semble pas être inspirée d'un phénomène naturel.

Une telle classification ne semble cependant pas très utile et est parfois difficile à réaliser. En effet, il existe de nombreuses métaheuristiques récentes qu'il est difficile de les classer dans l'une des deux catégories. Certains se demandent par exemple si l'utilisation d'une mémoire dans la méthode Tabou n'est pas directement inspirée de la nature.

Une autre façon de classifier les métaheuristiques est de distinguer celles qui travaillent avec une population de solutions de celles qui ne manipulent qu'une seule solution à la fois. Les méthodes qui tentent itérativement d'améliorer une seule solution sont appelées méthodes de recherche locale ou méthodes à trajectoire. La méthode Tabou, le Recuit Simulé et la Recherche à Voisinages Variables sont des exemples typiques de méthodes à trajectoire. Ces méthodes construisent une trajectoire dans l'espace des solutions en tentant de se diriger vers la solution optimale.

Les métaheuristiques à base de population cherchent à améliorer, au fur et à mesure des itérations, une population de solutions. L'exemple le plus connu des méthodes qui travaillent avec une population de solutions est l'algorithme génétique.

Les métaheuristiques peuvent également être classées selon leur manière d'utiliser la fonction objectif. Étant donné un problème d'optimisation consistant à minimiser une fonction f sur un espace S de solutions, certaines Métaheuristiques dites statiques travaillent directement sur f alors que d'autres, dites dynamiques, font usage d'une fonction g obtenue à partir de f en ajoutant quelques composantes qui permettent de modifier la topologie de l'espace des solutions, ces composantes additionnelles peuvent varier durant le processus de recherche.

Des chercheurs préfèrent classifier les métaheuristiques en fonction du nombre de structures de voisinages utilisées. Étant donné qu'un minimum local relativement à un type de voisinage ne l'est pas forcément pour un autre type de voisinage, il peut être intéressant d'utiliser des métaheuristiques basées sur plusieurs types de voisinages.

Certaines métaheuristiques font usage de l'historique de la recherche au cours de l'optimisation, alors que d'autres n'ont aucune mémoire du passé. Les algorithmes sans mémoire sont en fait des processus markoviens puisque l'action à réaliser est totalement déterminée par la situation courante. Les métaheuristiques qui font usage de l'historique de la recherche peuvent le faire de diverses manières. On différentie généralement les méthodes

ayant une mémoire à court terme de celles qui ont une mémoire à long terme.

1.4 Conclusion

Les métaheuristiques constituent une classe de méthodes approchées adaptables à un très grand nombre de problèmes combinatoires et de problèmes d'affectation sous contraintes, mais il est difficile de comparer entre ces différentes méthodes pour deux raisons :

- il faut affiner les paramètres intervenant dans ces méthodes avec la même rigueur.
- la qualité de la solution dépend du temps d'exécution.

Cependant, il est important de constater que ces métaheuristiques ont révélé leur grande efficacité pour fournir des solutions approchées de bonne qualité, à des problèmes de taille importante, dans un temps plutôt raisonnable.

Chapitre 2

optimisation par essaims d'abeilles

Sommaire

Les métaheuristiques sont des méthodes essentielles pour résoudre les problèmes NP-Difficile, elles peuvent être appliquées sur de nombreux problèmes. De leur part, les métaheuristiques peuvent être classées en deux catégories :les métaheuristiques à base de solution unique et les métaheuristiques à base de population de solutions.

Une métaheuristique à base de solution unique manipule une seule solution durant la procédure de recherche. Elle lance la recherche avec une seule solution et essaye d'améliorer sa qualité au cours des itérations. Cependant, une métaheuristique à base de population de solutions manipule un ensemble de solutions parallèlement. Elle lance la recherche avec une population de solutions et essaye d'améliorer leurs qualités au cours des itérations dans le but de fournir la ou les meilleures solutions trouvées. Nombreuses métaheuristiques sont inspirées de systèmes naturels, par exemple : le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes génétiques qui sont inspirées des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence par essaim comme l'algorithme BSO, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles.
Dans ce chapitre on va utiliser une méthode qui appartient à la classe des méthodes inspiré du comportement des animaux qui est l'algorithme d'optimisation par essaims d'abeilles.

2.1 L'intelligence en essaim

Le domaine de l'intelligence en essaim concerne l'intelligence collective émergée issue de simples agents agissant ensemble. Les insectes sociaux, comme par exemple les fourmis et les abeilles, sont la source d'inspiration. Comment l'interaction de nombreuses fourmis simples peut-elle créer des nids et des structures sociales complexes ? Est-il possible d'utiliser des techniques similaires dans les algorithmes informatiques et la robotique ?

Les sociétés de bogues ont une capacité remarquable à résoudre les problèmes de manière très souple (la colonie s'adapte aux changements brusques de l'environnement) et robuste (la colonie continue de fonctionner même lorsque certains individus ne parviennent pas à accomplir leurs tâches). Les problèmes quotidiens résolus par une colonie sont nombreux et de nature très variée : recherche de nourriture, construction du nid, répartition du travail et allocation des tâches entre les individus, ... etc.

En fait, ce comportement collectif, souvent complexe, est le fruit d'une agrégation de comportements individuels dictés par des règles très simples. Il présente un modèle de travail auto-organisé, basé sur une logique décentralisée, fondé sur la coopération d'unités ne disposant que d'informations locales.

Les ethnologues l'ont observé il y a longtemps, les programmeurs et les ingénieurs ont eu du mal à s'y approprier. Ce dernier pourrait transformer des modèles de comportement collectif de bugs sociaux en méthodes puissantes de conception d'algorithmes d'optimisation combinatoire. Un nouveau domaine de recherche est apparu. Il a pour but de transformer les connaissances des ethnologues sur les capacités collectives de résolution de problèmes par les bugs sociaux, en techniques efficaces de résolution de problèmes tout en offrant un haut degré de flexibilité et de robustesse : il s'agit d'un renseignement en essaim.

Parmi les techniques d'intelligence en essaim, certaines sont arrivées à maturité. Les algorithmes de contrôle et d'optimisation inspirés notamment des modèles de recherche collective de l'alimentation chez les fourmis, ont connu un succès inattendu. Ces techniques sont utilisées dans plusieurs domaines d'application tels que l'informatique industrielle, la robotique, l'analyse de données, le réseau de télécommunication et l'optimisation combinatoire [10].

2.2 L'algorithme d'optimisation par essaims d'abeilles (bees swarm optimization)

En 1946, Karl Von Fris [11] a décodé le langage des abeilles. Il a observé que c'est par la danse qu'une abeille communique avec ses semblables lors de son retour dans la ruche, la distance, l'orientation et la richesse de la source de nourriture.
Les abeilles d'une même colonie visitent plus d'une douzaine d'aires d'exploitation potentielles. Mais la colonie concentre ses efforts de récolte sur un petit nombre d'entre elles, les plus riches et les plus faciles d'accès. En outre, de nombreuses observations semblent indiquer qu'une colonie peut remplacer rapidement l'exploitation d'une source par une autre.
Seely, Camazine et Sneyd, en 1991 [12], ont montré que lorsqu'une colonie d'abeilles a le choix entre l'exploitation de deux sources de nourriture où la concentration en sucre est très inégale (1 M et 2,5 M respectivement) et situé de manière diamétralement opposée à la ruche, l'un au nord et l'autre au sud, la colonie va concentrer son effort de récolte sur les plus riches. Nous expliquons cela par le fait que l'abeille suit l'abeille qui fait la danse la plus vigoureuse, donc celle qui indique la place de la source de nourriture la plus riche [13].
La méta-heuristique "optimisation par essaim d'abeilles" ou BSO s'inspire du comportement décrit ci-dessus. Il repose sur un comportement d'essaim d'abeilles artificielles coopérant pour résoudre un problème. BSO a été développée au sein du laboratoire LRIA de l'USTHB [1]. Le choix de cette méta-heuristique est motivé par les bons résultats fournis par son adaptation à des domaines comme la résolution du problème SAT et la recherche d'information à grande échelle [13, 14]. Le principe de cette approche repose sur l'observation du comportement des abeilles et plus exactement sur leur processus de recherche de la nourriture. Notons que BSO consiste à rechercher des points du champ d'investigation des abeilles intéressants pour leur nourriture

2.2.1 Analyse du comportement des abeilles

Seeley a décrit dans [1] le comportement des abeilles pour la recherche de la nourriture suite à des expériences entreprises sur des abeilles. La figure 1 illustre les expériences conduites pour comprendre un tant soit peu le comportement social des abeilles.

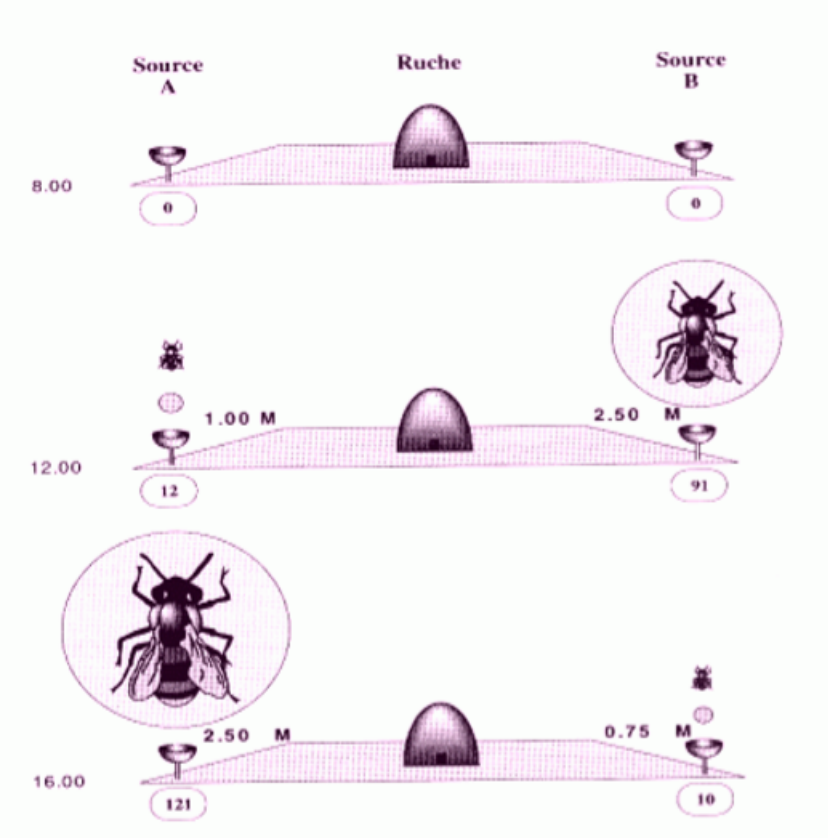

FIGURE 2.1 – Expérience de Seely sur le comportement des abeilles [1]

Deux sources de nourritures A et B sont placées de part et d'autre d'une ruche. A 8h00 du matin, les deux sources sont vides : aucun nectar n'est mis dans les récipients. On observe qu'aucune abeille ne se dirige vers ces sources. A 12h00, la source A est placée à 1mètre de la ruche et une quantité de 12 unités de nectar est mise dans le récipient correspondant. Par ailleurs, la source B est placée à 2,50 mètres et la quantité placée dans le récipient correspondant est égal à 91 unités de nectar. Après quelques instants, on observe qu'une abeille se dirige vers la source B qui est plus éloignée mais plus riche en nourriture et se met à danser vigoureusement. A 16h00, la source A est placée à 2,50 mètres de la ruche et une quantité de 121 unités de nectar est mise dans le récipient correspondant. Par ailleurs, la source B est placée à 0,75 mètres et la quantité placée dans le récipient correspondant est réduite à 10 unités de nectar. Après quelques instants, le même constat est remarqué, à savoir : on observe qu'une abeille se dirige vers la source A qui est plus éloignée mais plus riche en nourriture. Et de plus, la dance se fait avec plus de vigueur. Les résultats de l'expérience peuvent être résumés comme suit :

1- initialement les abeilles partent de leur ruche à la recherche de la nourriture, du nectar plus précisément.

2- Dès qu'une abeille trouve la nourriture, elle revient à la colonie. Lors de son retour elle réalise la distance et la direction entre la ruche et la source de nourriture ainsi que la quantité de cette dernière.

3- Chaque abeille communique ses propres informations (la distance, la direction et la quantité de nourriture) à ses congénères en effectuant une danse. Il existe plusieurs types de danses permettant d'indiquer les informations évoquées ci -dessus.
4- la colonie exploite par la suite, l'endroit où il y a le plus de nourriture, indiqué par l'abeille qui effectue la danse la plus vigoureuse et qui est la plus proche de la ruche.
5- Le même processus est réitéré à partir de cet endroit.

2.2.2 Principe de l'algorithme BSO

Le processus de recherche est initialisé par une solution référence appelée Sref représentant l'abeille éclaireuse. Elle est choisie aléatoirement ou à l'aide d'une heuristique et détermine la zone à exploiter appelée SearchArea et qui est constituée d'un ensemble de solutions possibles. Chaque point ou solution de cet espace est affecté à une abeille. Après cela chaque abeille artificielle commence à explorer la région qui lui a été affectée pour déterminer la meilleure solution locale. Elle stocke la solution obtenue dans une table appelée « Dance » et qui est visible de toutes les autres abeilles, c'est une façon de simuler la danse des abeilles réelles. Une fois que toutes les solutions sont déposées dans la table danse, la meilleure solution est sélectionnée (qui simule la danse la plus vigoureuse) pour devenir la solution de référence pour la prochaine itération. Les solutions de référence sont insérées dans une liste taboue afin d'éviter d'explorer des régions qui ont été déjà visitées. Le processus global est illustré dans la figure 2.2

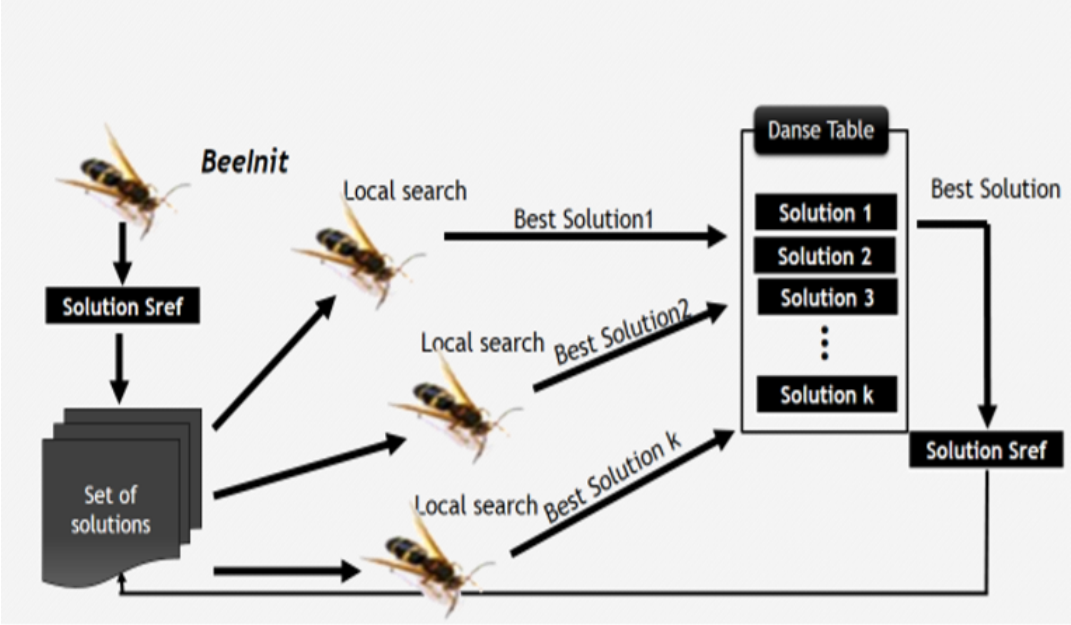

FIGURE 2.2 – Architecture de BSO [1]

2.2.2.1 Pseudo Code de l'algorithme BSO :

L'Algorithme 2.1 résume les étapes Principales d'algorithme d'optimisation par essaims d'abeilles

Algorithm 2.1 Schéma d'algorithme de BSO

1: $Sref \leftarrow$ La solution trouvée par InitBee.
2: **tantque** $i <$ **Max-Iter** et ne pas arrêter **faire**.
3: Insérer $Sref$ dans **taboo list**.
4: SearchArea(Sref).
5: Attribuer une solution de SearchArea à chaque bee.
6: **pour** chaque bee k **faire**.
7: Built-Search-Area(bee_k).
8: stocker le résultat dans la table Dance.
9: **fin pour**
10: choisie la nouvelle solution de référence $Sref$.
11: **fin tantque**

- La table Dance :

  Comme la plupart des méta-heuristique, BSO travaille sur une solution qu'il essaye d'améliorer à chaque itération. Une table contenant les meilleures solutions, appelée Dance, est utilisée. Nous avons opté à organiser cette table sous forme de tas, ainsi à chaque itération la racine du tas est choisie pour le traitement, suite à cela, les meilleures solutions trouvées par les abeilles à la fin de l'itération sont insérées dans la table.

- Algorithme de recherche :

  à chaque itération on essaye d'améliorer une solution initiale. L'itération commence par générer des solutions équidistantes de la solution initial, et pour chaque solution générée on fait une recherche locale. Ensuite, chacune des solutions trouvées localement est insérées dans la table Dance cité précédemment. L'itération suivante fera le même traitement en commençant par la meilleure solution de Dance. Cela est répété jusqu'à ce qu'on arrive à la solution optimale ou à une condition d'arrêt, nombre maximum d'itération atteint par exemple.

2.3 Conclusion

Dans ce chapitre, une nouvelle méta-heuristique appelée « Essaim d'abeilles » a été proposée. Elle s'inspire du comportement des abeilles réelles, dont le principe est de récolter le nectar des sources d'accès les plus faciles tout en privilégiant les plus riches. La danse des abeilles, qui est leur seul moyen de communication, est probablement le secret de leur efficacité puisque c'est à travers elle qu'une abeille indique à ses semblables une source de nourriture et sa qualité. Il entraîne très rapidement l'essaim pour concentrer ses efforts de recherche sur la source la plus riche.

Dans ce travail, nous nous intéressons à un problème NP-Difficile, à savoir, le problème de l'arbre dominant (DTP). Dans le chapitre suivant, nous adaptons la méta-heuristique BSO pour résoudre le problème d'arbre dominant.

Chapitre 3

Problème de l'arbre dominant

Sommaire

Un mathématicien confronté à un problème de la vie réelle s'empresse de traduire celui-ci en un problème mathématique, qu'il peut alors tenter de résoudre. Dans sa boîte à outils mathématiques, les graphes peuvent s'avérer fort utiles. [15]

Les graphes sont un outil de modélisation puissant et très intuitif. Ils sont naturellement utilisés pour représenter des réseaux de transport, de communication, et plus généralement des flux de matières ou d'informations. Ils sont également utilisés pour modéliser des problèmes dans lesquels des objets sont en relation, les sommets du graphe représentant ces objets et les arêtes (ou arcs si une orientation est considérée) les relations entre ces objets. [16]

Le problème d'optimisation considéré est modélisé par les outils de la théorie des graphes ensuite une solution est cherchée qui est généralement un algorithme. Parmi les problèmes modélisés par la théorie des graphes on cite : le problème d'affectation, de flot maximum ou minimum, de transport, de voyageur de commerce, de l'arbre dominant etc. [17]

Dans ce chapitre nous présentons des généralités sur les graphes et le problème de l'arbre dominant liées au travail ainsi que les problèmes connectés au problème DT.

3.1 Généralités sur les graphes

On fait généralement remonter la naissance de la Théorie des Graphes au célèbre problème des ponts de Königsberg (aujourd'hui Kaliningrad) qui passionnait la bourgeoisie prussienne du 20ème siècle : La Ville de Königsberg, sur la Pregel, était pourvue de 7 ponts et la question était de savoir si l'on pouvait imaginer une promenade dans la ville qui emprunterait chacun des 7 ponts une fois et une seule pour revenir à son point de départ. [18]

3.1.1 Graphe

De façon plus formelle, un graphe est défini par un couple $G = (S, A)$ tel que :

— **S** est un ensemble fini de sommets

— **A** est un ensemble de couples de sommets $(s_i, s_j) \in S^2$.

graphe orienté

Un graphe orienté G est une paire (S, A) où S est un ensemble fini de sommets et A est une relation binaire sur Set constitue l'ensemble des arêtes de G. Dans ce cas, on parlera d'arc plutôt que d'arête. Un couple (s_i, s_j) est appelé un arc, et est représenté graphiquement par $s_i \rightarrow s_j$

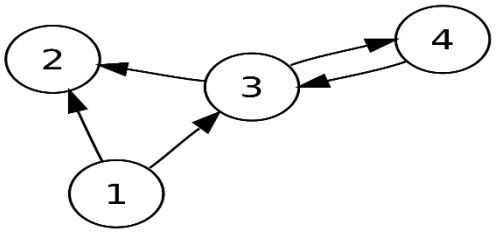

FIGURE 3.1 – Un graphe orienté

Graphe non orienté

Un graphe non-orienté G est une paire (S, A) où S est un ensemble fini de sommets et A est un ensemble de couple(s) non ordonné(s) de sommets. De plus, les boucle sont interdites dans les graphes non-orientés. Une paire $\{s_i, s_j\}$ est appelée une arête, et est représentée graphiquement par $s_i{}^{-}s_j$

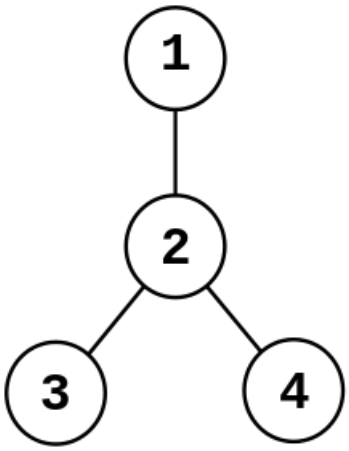

FIGURE 3.2 – Un graphe non orienté

Terminologie

— L'**ordre** d'un graphe est le nombre de ses sommets.

— Une **boucle** est un arc ou une arête reliant un sommet à lui-même.

— Un graphe non-orienté est dit **simple** s'il ne comporte pas de boucle, et s'il ne comporte jamais plus d'une arête entre deux sommets. Un graphe non orienté qui n'est pas simple est un **multigraphe**.
Dans le cas d'un multi-graphe, A n'est plus un ensemble mais un multi-ensemble d'arêtes. On se restreindra généralement dans la suite aux graphes simples.

— Un graphe orienté est un **p-graphe** s'il comporte au plus p arcs entre deux sommets. Le plus souvent, on étudiera des 1-graphes.

— Un **graphe partiel** d'un graphe orienté ou non est le graphe obtenu en supprimant certains arcs ou arêtes.

— **Arbres Couvrants** est un graphe partial est sans Cycle.

— Un **sous-graphe** d'un graphe orienté ou non est le graphe obtenu en supprimant certains sommets et tous les arcs ou arêtes incidents aux sommets supprimés.

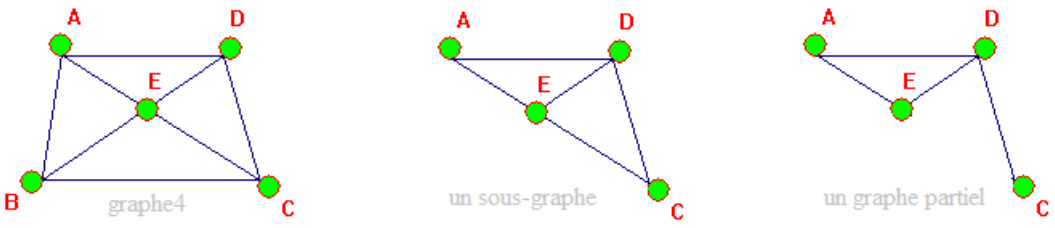

FIGURE 3.3 – exemple d'un graphe partiel et sous-graphe

— Un graphe orienté est dit **élémentaire** s'il ne contient pas de boucle.

— Un graphe orienté est dit **complet** s'il comporte une arête (s_i, s_j) pour toute paire de sommets différents $s_i, s_j \in S^2$.

3.1.2 Notion d'adjacence entre sommets

Dans un graphe non orienté, un sommet s_i est dit adjacent à un autre sommet s_j s'il existe une arête entre s_i et s_j .
L'ensemble des sommets adjacents à un sommet s_i est défini par :
$adj(s_i) = \{s_j / (s_i, s_j) \in A \text{ ou } (s_j, s_i) \in A\}$

— Dans un graphe orienté, on distingue les sommets successeurs des sommets prédécesseurs :
$succ(s_i) = \{s_j / (s_i, s_j) \in A\}$
$pred(s_i) = \{s_j / (s_j, s_i) \in A\}$

3.1.3 Notion de degré d'un sommet

— Dans un graphe non orienté, le **degré** d'un sommet est le nombre d'arêtes incidentes à ce sommet (dans le cas d'un graphe simple, on aura $d^0(s_i) = |adj(s_i)|$.

— Dans un graphe orienté, le **demi-degré extérieur** d'un sommet s_i, noté $d^{0+}(s_i)$, est le nombre d'arcs partant de s_i (dans le cas d'un 1-graphe, on aura $d^{0+}(s_i) = |succ(s_i)|$). De même, le **demi-degré intérieur** d'un sommet s_i, noté $d^{0-}(s_i)$, est le nombre d'arcs arrivant à s_i (dans le cas d'un 1-graphe, on aura $d^{0-}(s_i) = |pred(s_i)|$).

3.1.4 Cheminements et connexités

3.1.4.1 Notions de chemin, chaine, cycle et circuit

Dans un graphe orienté, un **chemin** d'un sommet u vers un sommet v est une séquence $< s_0, s_1, s_2, \ldots, s_k >$ de sommets tels que $u = s_0$, $v = s_k$, et $(s_{i-1}; s_i) \in A$ pour $i \in [1..k]$.
La **longueur** du chemin est le nombre d'arcs dans le chemin, c'est-à-dire k. On dira que le chemin contient les sommets $s_0, s_1, ..., s_k$, et les arcs $(s_0, s_1), (s_1, s_2), \ldots, (s_{k-1}, s_k)$.
S'il existe un chemin de u à v, on dira que v est accessible à partir de u.
Un chemin est **élémentaire** si les sommets qu'il contient sont tous distincts.
Dans un graphe orienté, un chemin $< s0, s1, \ldots, sk >$ forme un **circuit** si $s0 = sk$ et si le chemin comporte au moins un arc ($k \geq 1$). Ce circuit est **élémentaire** si

en plus les sommets $s1, s2, \ldots, sk$ sont tous distincts. Une boucle est un circuit de longueur 1.

Considérons par exemple le graphe orienté dans la figure 3.4 :

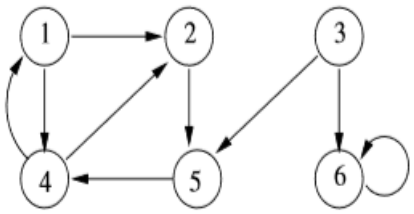

FIGURE 3.4 – chemin, circuit élémentaire et non élémentaire

Un chemin élémentaire dans ce graphe est $< 1; 4; 2; 5 >$.

Un chemin non élémentaire dans ce graphe est $< 3; 6; 6; 6 >$.

Un circuit élémentaire dans ce graphe est $< 1; 2; 5; 4; 1 >$.

Un circuit non élémentaire dans ce graphe est $< 1; 2; 5; 4; 2; 5; 4; 1 >$.

On retrouve ces différentes notions de cheminement dans les graphes non orientés. Dans ce cas, on parlera de **chaine** au lieu de chemin, et de **cycle** au lieu de circuit. Un graphe sans cycle est dit **acyclique**.

3.1.4.2 Notions de connexité

— **Un graphe non orienté connexe**

On dit Un graphe non orienté est connexe :

- si chaque sommet est accessible à partir de n'importe quel autre.

- si pour tout couple de sommets distincts $(s_i, s_j) \in S^2$, il existe une chaine entre s_i et s_j.

FIGURE 3.5 – un graphe non oriente n'est pas connexe

Car il n'existe pas de chaîne entre les sommets a et e. Mais, le sous-graphe est défini par les sommets $\{a, b, c, d\}$ est connexe.

— Une **Composante connexe d'un graphe non orienté**

Composante connexe d'un graphe non orienté G est un sous-graphe G' de G

qui est connexe et maximal(c'est-à-dire qu'aucun autre sous-graphe connexe de G ne contient $G0$).

— **Un graphe orienté fortement connexe**

Un graphe orienté est fortement connexe :

- si chaque sommet est accessible à partir de n'importe quel autre.

- si pour tout couple de sommets distincts $(s_i, s_j) \in S^2$, il existe un chemin de s_i vers s_j , et un chemin de s_j vers s_i.

FIGURE 3.6 – le graphe gauche fortement connexe, le droite ne l'est pas

— Une **composante fortement connexe** d'un graphe orienté G est un sous-graphe G' de G qui est fortement connexe et maximal (c'est à dire qu'aucun autre sous-graphe fortement connexe de G ne contient G').

3.1.4.3 Distance

Soit G un graphe et s, s' deux sommets. La **distance** entre s et s' est la plus petite longueur d'un chemin d'extrémité s et s'.

Dans un graphe orienté, la distance de s vers s' est la plus petite longueur d'un chemin orienté allant de s à s'.

Le diamètre $D(G)$ du graphe G est la plus grande des distances dans G :

$D(G) = \max_{x,y \in} V$ d(x,y).

3.1.5 Arbres

On appelle arbre tout graphe connexe sans cycle.

- une forêt est Un graphe sans cycle mais non connexe.

- Une feuille ou sommet pendant est un sommet de degré 1.

Quelques propriétés :

1. G est un arbre,
2. G est sans cycle et connexe,
3. G est sans cycle et comporte $n-1$ arêtes,
4. G est connexe et comporte $n-1$ arêtes,

5. chaque paire u, v de sommets distincts est reliée par une seule chaîne simple (et le graphe est sans boucle).

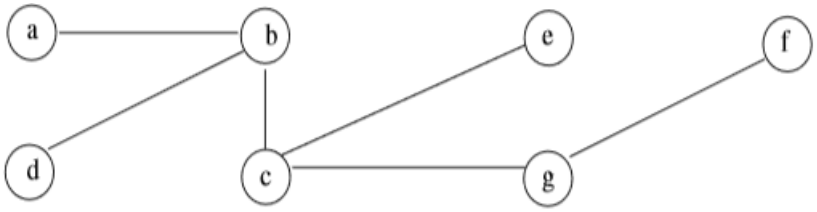

FIGURE 3.7 – cet graphe est un arbre

3.2 Quelques problèmes principaux

3.2.1 Ensemble dominant

Un sous-ensemble de sommets, $D \subseteq V$, est un ensemble dominant du graphe $G = (V, E)$
- si chaque sommet de G est soit un membre de D, soit adjacent à un sommet de D.
- si D induit un sous-graphe connexe, il est appelé ensemble dominant connexe **(connected dominating set,CDS)**[19].

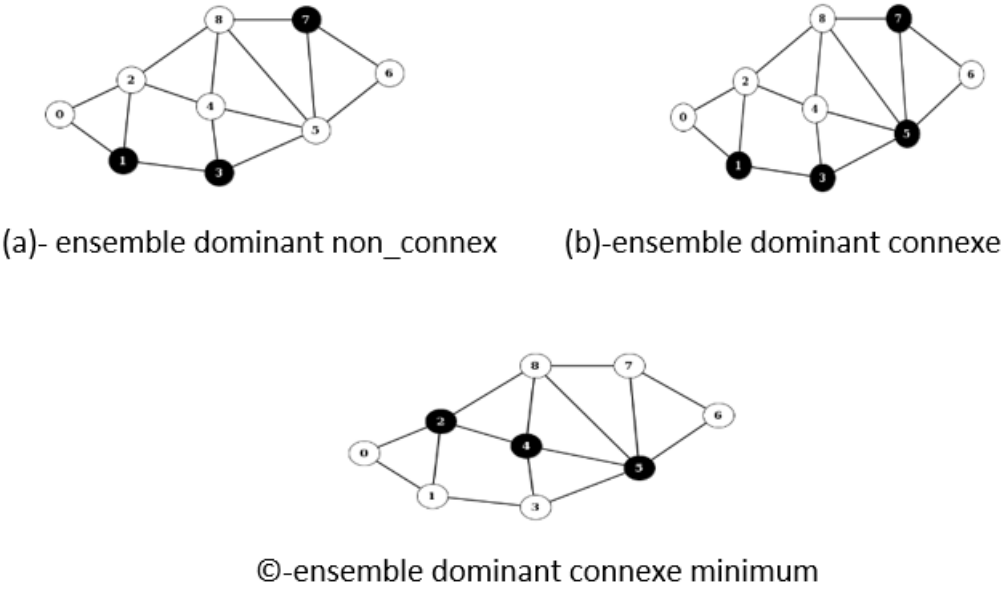

FIGURE 3.8 – Types ensembles dominants

3.2.1.1 La cardinalité

Le cardinal minimum d'un ensemble dominant connexe, noté $\gamma c(G)$, est appelé le nombre de dominance connexe de G.

3.2.2 Un CDS (connected dominating set)

Un CDS ayant une taille égale au nombre de domination connexe est dit ensemble dominant connexe minimum ou (MCDS) .
Plus formellement, un ensemble dominant d'un graphe $G = (V, E)$ est un sous-ensemble $D \subseteq V$ tel que $T(D) \cup D = V$.Autrement dit,$\forall v \in V\,; (T(v) \subseteq \{v\}) \cap D6 = \emptyset$.Un ensemble dominant D est dit connexe si le sous-graphe $(D, E(D))$, qu'il induit, est connexe, avec $E(D) = \{\{i, j\} \in E | i \in D, j \in D\}$.

3.2.2.1 Poids d'un graphe

Le poids(où coût) d'un graphe est la somme des poids des arêtes du graphe

3.2.3 Arbre de poids minimal (MST)

Soit le graphe $G = (V, E)$ avec un poids associé à chacune de ses arêtes. On veut trouver, dans G, un arbre maximal $A = (V, F)$ de poids total minimum.
Algorithme kruskal :

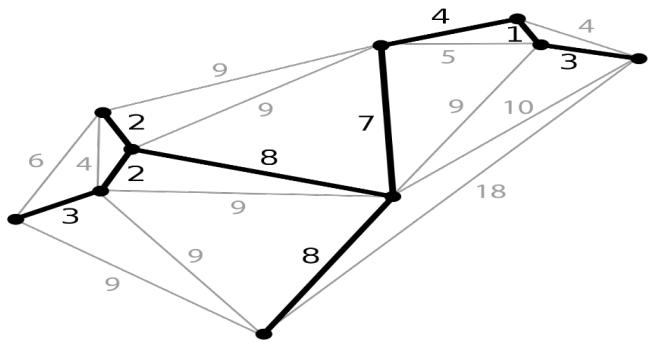

FIGURE 3.9 – arbre couvrant de point minimal

— **Données :**
Un graphe composé de N sommets est donné par la liste de ses arêtes dans l'ordre de leur poids croissant. Soit T l'arbre recherché.

— **Méthode :**
La liste des arêtes est lue en commençant par l'arête de plus faible poids, notée u_1.
Au départ $T = \{\quad u_1\}$.
A l'étape k l'arête $\quad u_k$ est lue. Si elle ne forme pas de cycle avec les arêtes de T alors elle est retenue. L'algorithme s'arrête quand le nombre d'arêtes de T est égal à $N - 1$.

— **ALGORITHME :**

Soit $G = (X, U)$ avec $|X| = N$ et $|U| = M$.

1) Classer les arêtes par ordre de longueur croissante (u_1 , u_2 , u_3 ,...., u_M)

2) $T = 1, i = 1$

3) Si $T +$ u_i est sans cycle ajouter u_i à T.

$T = T + \{ u_1\}$

Sinon aller en 4.

4) Incrémenter i de 1.

Si $|T| = N - 1$ Stop, sinon retourner en 3.

3.2.4 Couverture par sommets(transversale)

Une couverture par sommets d'un graphe G est un ensemble C de sommets tel que chaque arête de $G = (V, E)$ est incidente à au moins un sommet de C, ie un sous-ensemble de sommets $S \subseteq V$ tel que pour chaque arête (u, v) de G on a $u \in S$ ou $v \in S$. On dit que l'ensemble C couvre les arêtes de G. La figure suivante montre des exemples de couvertures des sommets de deux graphes.

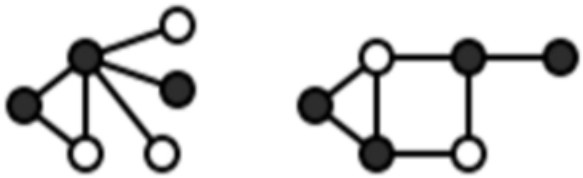

FIGURE 3.10 – Example de couverture par sommets

Une couverture minimale par sommets est une couverture des sommets de taille minimale.

— La recherche d'un ensemble transversal minimum (minimum vertex cover problem) est le problème d'optimisation :
Données : Graphe G
Question : Le plus petit nombre k tel que G a un transversal de taille k

— Problème de décision correspondant (vertex cover problem) :
Données : Graphe G et un entier positif k
Question : Est-ce que G contient un transversal de taille k ?

Si un ensemble de sommets S est un transversal, son complément est un stable (ou ensemble indépendant).

3.2.5 Couverture par arêtes

La couverture par arêtes d'un graphe G est un ensemble des arêtes C tel que chaque sommet de G est incident avec au moins une arête en C. L'ensemble C est censé recouvrir les sommets de G. La figure 3.10 montre des exemples de couverture des arêtes de deux graphes.

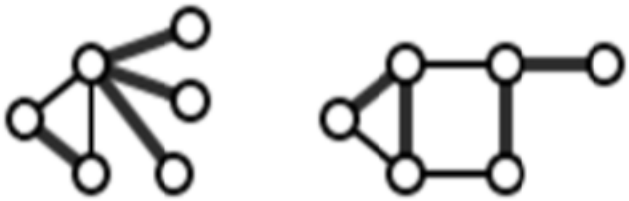

FIGURE 3.11 – exemples de couverture par arête

3.2.6 La coloration

En théorie des graphes, la coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente[20]

— **Une coloration des sommets** d'un graphe G est une affectation de couleurs aux sommets telle que chaque arête de G ait ses deux extrémités de couleur différente.

On cherche généralement à déterminer une coloration utilisant aussi peu de couleurs que possible.

– **le nombre chromatique** de G est le plus petit nombre de couleurs nécessaire pour colorer les sommets d'un graphe G et est noté (G).[21]

— **Une coloration des arêtes** d'un graphe G est une affectation de couleurs aux arêtes telle que les arêtes ayant une extrémité en commun soient de couleur différente.

On cherche généralement à déterminer une coloration utilisant aussi peu de couleurs que possible. – **l'indice chromatique** est le plus petit nombre de couleurs nécessaires pour colorer les arêtes d'un graphe G et est noté q(G).[21]

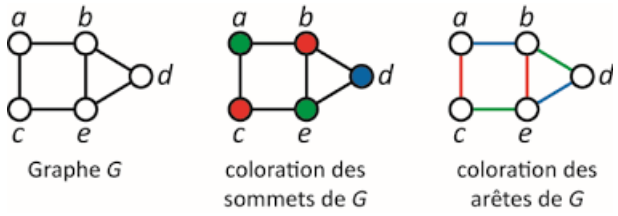

FIGURE 3.12 – la coloration des sommets et des arrêts de graphe

Algorithme de Welsh et Powell

1. Repérer le degré de chaque sommet.
2. Ranger les sommets par ordre de degrés décroissants (dans certains cas plusieurs possibilités).
3. Attribuer au premier sommet (A) de la liste une couleur.
4. Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.
6. Continuer jusqu'à ce que la liste soit finie.
7. Prendre une deuxième couleur pour le premier sommet (D) non encore colorié de la liste.
8. Répéter les operations 4 à 6.
9. Continuer jusqu'à avoir coloré tous les sommets.

3.3 Problème de l'arbre dominant

3.3.1 Description de problème d'arbre dominant

Dans [22]. Ce problème est défini comme suit :

Soit $G = (V, E)$ un graphe non orienté, connecté, pondéré, où V désigne l'ensemble des sommets et E désigne l'ensemble des arêtes. A chaque arête $e \in E$, un poids non négatif w_e est attribué.

Un arbre $T = (V(T), E(T))$ du graphe G est appelé arbre dominant :

- si chaque sommet $v \in V$ qui n'est pas dans T est adjacent à un sommet en T.

ou autrement dit :

Un arbre dominant $T = (V(T), E(T))$ de G avec $V(T) \subset V$ et $E(T) \subset E$ est un graphe connexe acyclique où chaque nœud $v \in V \backslash V(T)$ est adjacent à un nœud en $V(T)$.

Le poids d'un arbre T est défini comme suit :

$$\sum_{e \in E(T)} c_e \tag{3.1}$$

tel que $c_e \in R+$ désigne le poids d'une arête e $\in E$.[22]
Le problème de l'arbre dominant (DTP) est de construire un arbre dominant T du graphe G avec un poids minimal.
Le DTP a plusieurs applications dans la conception de réseau et le routage réseau. La multidiffusion est un exemple donné dans [22], dont le but est la livraison simultanée des mêmes données à un groupe d'ordinateurs de destination.
Les serveurs sont connectés par une structure de réseau arborescente T, et tous les autres ordinateurs sont à un bond d'un serveur. Si les poids représentent le coût ou l'énergie pour transmettre des données d'un serveur à l'autre, la somme des poids des arêtes en T équivaut à un coût global pour transmettre des données d'un serveur à l'autre.
Les premières approches métaheuristiques pour résoudre le DTP ont été proposées dans [23], où les auteurs ont mis en œuvre deux techniques d'intelligence d'essaim : la colonie d'abeilles artificielles et l'optimisation des colonies de fourmis. L'algorithme d'optimisation des colonies de fourmis a produit de meilleurs résultats sur la plupart des grandes instances, mais il était plus lent que l'algorithme d'optimisation des colonies d'abeilles artificielles.[23]
La figure 3.14 montre un exemple du problème DT.

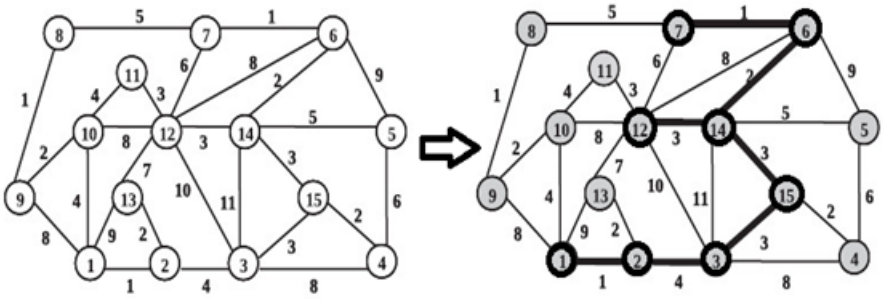

FIGURE 3.14 – exemple d'un problème d'arbre dominant DT

Formulation du problème

Soit un arbre dominant T de G représenté par un vecteur d'incidence $x \in \{0,1\}^{|E|}$, où $x_{uv} = 1$ si $uv \in E(T)$ et $x_{uv} = 0$, sinon.
Définir les variables binaires $y_v \in \{0,1\}$ pour tout $v \in V$, où $y_v = 1$ si $v \in V(T)$, et $y_v = 0$, sinon. Soit $E(S)$ désigne l'ensemble des arêtes avec les deux points d'extrémité dans $S \subset V$ et soit $N(v) = \{u \mid uv \in E\}$ représente le voisinage d'un nœud v. Un modèle

exponentiel [22] pour ce problème est comme suit :

$$(P) \qquad \min_{x,y} \sum_{uv \in E} c_{uv} x_{uv} \tag{3.2}$$

$$s.t. \quad \sum_{uv \in E} x_{uv} - \sum_{v \in V} y_v = -1, \tag{3.3}$$

$$\sum_{uv \in E(s)} x_{uv} \leq |S| - 1, \forall\ S \subset V, \tag{3.4}$$

$$y_u + y_v \geq 2x_{uv}, \forall uv \in E, \tag{3.5}$$

$$\sum_{u \in N(v)} y_u \geq 1, \forall\ v \in V, \tag{3.6}$$

La contrainte 3.3 établit qu'un arbre T de $\sum_{v \in V} y_u$ nœuds ont $\sum_{v \in V} y_v - 1$ arêtes, tandis que la contrainte 3.4 évite les cycles induits par les arêtes de $E(T)$. Contrainte 3.5, garantit que si $uv \in E(T)$, les noeuds u et v appartiennent à $V(T)$. Par les contraintes de domination 3.6, tout nœud v dans V a un voisin dans T.

3.4 Problèmes connexes au problème DTP

3.4.1 CDSP(The Connected Dominating Set Problem)

L'un des problèmes associée au DTP est le problème de l'ensemble dominant connecté (CDSP), Ce problème peut être formulé comme suit :
Pour un graphe, trouver un ensemble dominant connecté de taille minimale. Dans quelques variantes de ce problème les sommets peuvent avoir des poids et l'objectif est de minimiser la somme totale pondérée des sommets qui forment un ensemble dominant connecté [24]. Il convient de noter que dans le problème DTP, les poids sont associés aux arêtes et non pas aux sommets. [25]

3.4.2 TCP(Tree Cover Problem)

Le problème de couverture d'arbre (TCP) considère un graphe où chaque arête a un poids non négatif, le problème consiste à trouver un arbre T dans le graphe G avec poids minimal et qui représente une couverture par sommets de G c.-à-d. chaque arête de G a au moins une extrémité $\in T$. Le résultat alors est un arbre qui représente un ensemble dominant des arêtes. Contrairement au DTP où l'arbre_résultant est un ensemble dominant de sommets.

3.5 Conclusion

Dans ce chapitre, nous avons vu quelques notions de base de Théorie des Graphes et présenté le problème étudié de l'arbre dominant DTP.Nous nous basons beaucoup plus sur le livre [Shin et al., 2010]

Dans le chapitre qui suit, nous présentons notre approche proposée pour résoudre le problème d'arbre dominant à savoir, l'optimisation par essaim d'abeilles BSO

Chapitre 4

optimsation par essaims d'abeilles pour le problème d'arbre dominant

Sommaire

Dans ce chapitre, nous proposons un algorithme d'optimisation par essaims d'abeilles BSO pour résoudre le problème de l'arbre dominant. L'objectif est d'adapter cette méthode pour résoudre le problème DTP. Pour cela, nous présentons l'encodage de la solution proposé, ainsi que la description des étapes de l'algorithme.

4.1 Encodage de la solution

Le codage associe à une solution une structure de données. Elle se place généralement après une phase de modélisation du problème traité. Nous proposons de coder la solution comme une liste de trois éléments. Le premier élément est une liste contenant les sommets de l'arbre dominant. Le deuxième élément est la liste des sommets restants du graphe (qui n'appartiennent pas à l'arbre dominant). Le dernier élément de la liste est le coût total de la solution.

Exemple :

Supposant qu'on ait un réseau de 10 nœuds (sommets) (codés de 0 à 9), une solution faisable contient tous les sommets (figure 4.1) La solution est :

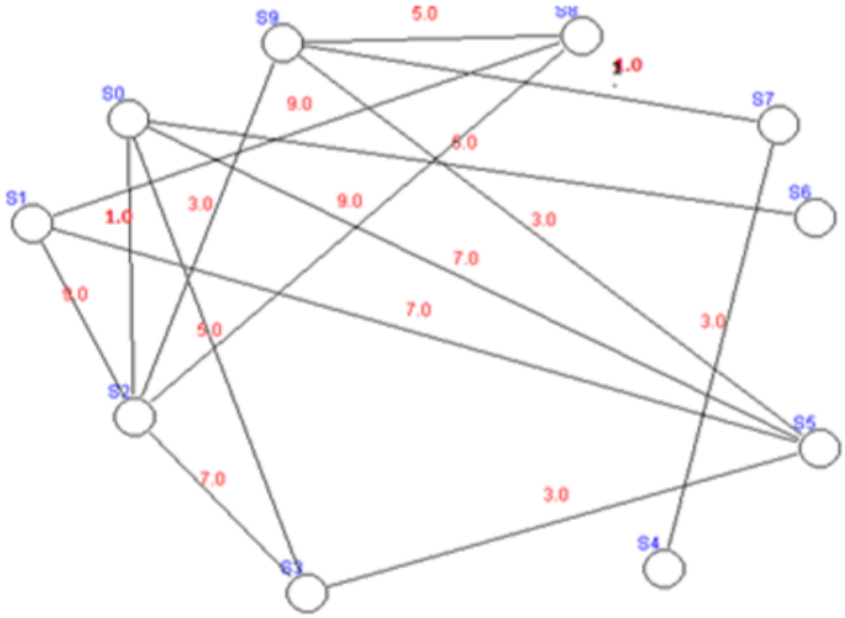

FIGURE 4.1 – exemple de solution faisable d'un graphe

v_s : $\{s_9, s_8, s_7, s_5, s_0\}$

v_non_s : $\{s_1, s_2, s_3, s_4, s_6\}$

cout : 12

v_s : représente les sommets qui composent l'arbre dominant

v_non_s : représente le rest des sommets de graphe (Qui n'appartient pas à la solution)

cout : représente le coût total d'une solution

4.2 Algorithme d'optimisation par essaim d'abeilles pour le DTP

Dans l'algorithme BSO la population est mise à jour en appliquant successivement la procédure de mutation, puis on utilise la recherche locale pour sortir de l'optimum local. Le pseudo code de l'algorithme est présenté dans Algorithme 4.1. L'algorithme de base BSO a été expliqué en détail dans le chapitre 2. Le schéma de mutation utilisé ici est BSO/best/1 car il donne de meilleures performances et converge rapidement vers les bonnes solutions.

4.2.1 Les principale étapes de l'algorithme

1-initialisation (préparation des structures et des entrées)

-lecture du fichier (les instances)

-mettre les valeur dans une structure List<arête>

note : une arête : sommet 1 et sommet 2 et ses poids(cout entre les deux sommets)

-construire les successeurs pour chaque sommet

2-generation d'une solution initiale(aléatoire)

-sélectionner un sommet

-sélectionner un de ces successeurs

-mettre les deux sommets dans une liste

-tant que l'arbre est non dominant

Ajouter un successeur de la liste de sommets choisis aléatoirement

Fin tant que

-mettre la solution dans Sref

3-génération des solutions voisines (bees)

-sélectionner l'arbre dominant de Sref et la mettre dans une liste

-supprimer un sommet de la liste aléatoirement

-permuter un sommet aléatoire qui n'appartient pas à l'arbre dominant avec le 1er

-tester si dominant et corrigé en cas de besoin

-mettre la solution (bee) dans une liste de solutions voisines

4-calculer le fitness(cout)

-calculer le cout pour chaque solution voisine

-mettre dans la table dance la meilleure solution parmi l'ensemble des bees (de cout minimal)

5 -condition d'arrêt

-Le nombre d' itérations (fixé manuellement)

6-Afficher la meilleure solution de table dance

Algorithme 4.1 Pseudo code de l'algorithme BSO

1 : Initialiser les paramètres (nombre de bees(voisines) et nombre max d'itérations)

2 : trouver une solution initiale (arbre dominant aléatoire) que nous appelons Sref

3 : tantque t $<$ Gmax faire

4 : générer les voisines (Bees) de la solution Sref (voisinage_1)

5 : tester si Bee est dominant et corriger en cas de besoin

6 : mettre dans la liste voisines

7 : calculer le cout pour chaque solution (Sref et voisines)

8 : mettre dans dance la meilleure solution courante (cout minimal)

9 : Sref $\leftarrow$ meilleure solution courante

10 : fin tantque

4.2.2 Réglage de paramètres

Le paramètres Nb_bees ont été fixés manuellement.

Nb_bees est le nombre de voisines générer dans chaque itération.

Le nombre de génération Gmax représente combien de fois il est nécessaire d'itérer le processus de recherche des solutions pour permettre à l'algorithme de converger vers une bonne solution. Si une recherche ne parvient pas à converger dans un certain nombre de génération, cela indique souvent que la recherche est devenue piégée dans un minimum local, alors Gmax est utilisé pour mettre n aux recherches qui n'ont pas réussi à converger après un nombre pratique de générations (Dans ce cas, nous choisissons dans la prochaine génération une solution aléatoire parmi les 5 meilleures Bees). Dans notre travail, nous avons fixé Gmax entre 10 et 500 générations. Cela dépend de la taille de problème. La taille de population NP dépendra aussi de la taille de problème, pour les petites instances 10-15 vecteurs sont suffisants pour atteindre une bonne solution, mais pour les grandes instances nous utilisons entre 100-500 vecteurs.

4.2.3 Création de solution initiale

Dans cette étape La population initiale est générée par plusieurs méthodes pour assurer une bonne diversité des combinaisons.

— La première méthode est de générer une population complètement aléatoire, le processus est comme suit :

On choisit un sommet s_1 aléatoire parmi la liste des sommets, puis on choisit aléatoirement un sommet s_2 parmi les successeurs et on construit un ensemble E. Si cet ensemble est dominant on arrête, sinon on choisit un sommet aléatoire dans E et on ajoute un des successeurs aléatoires dans E .On répète ce processus jusqu'à ce qu'on construit un ensemble dominant.

A partir de cet ensemble, on construit un arbre dominant.

L'algorithme de création d'une solution initiale aléatoire est noté (Algorithme 4.1)

Algorithm 4.1 Création d'une solution initiale aléatoire

1: Choisir un sommet aléatoire s_1
2: Choisir un sommet s_2 aléatoire parmi les successeurs de s_1
3: Ens_dom = $\{s_1, s_2\}$
4: Tester si Ens_dom est dominant
5: **tantque** Ens_dom non dominant **faire**
6: Choisir un sommet s_i aléatoire dans Ens_dom et on ajoute l'un des successeurs dans Ens_dom
7: **fin tantque**

Exemple :

Prenant le graphe suivant :

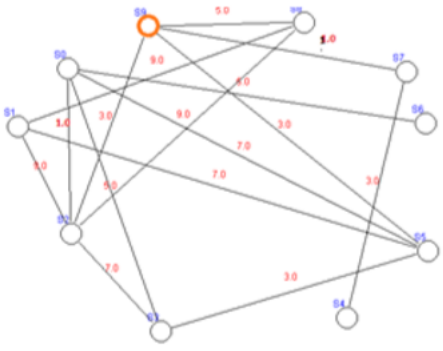

FIGURE 4.2 – choix aléatoire du 1er sommet (s_9) pour solution initiale

Supposant on choisit s_9 (figure 4.2) alors $S = [\{s_9\}]$, S n'est pas un ensemble dominant on continue la génération.

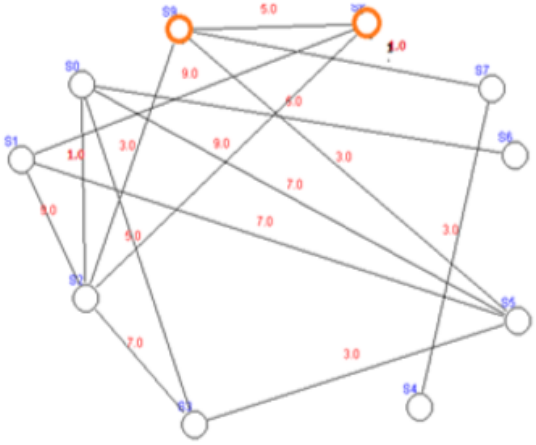

FIGURE 4.3 – choix aléatoire du 2ème sommet (s_8) pour solution initiale

On choisit s_8 aléatoirement parmi les successeurs de s_9 (figure 4.3) alors $S = [\{s_9, s_8\}]$, S n'est pas un ensemble dominant on continue.

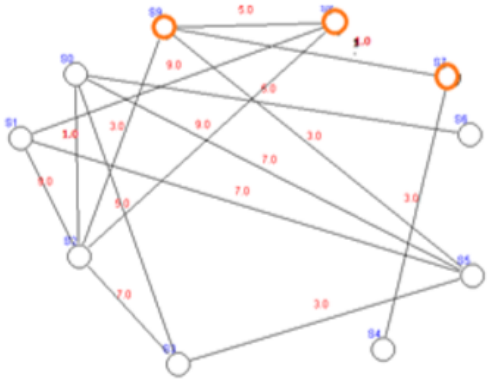

FIGURE 4.4 – choix aléatoire du 3ème sommet (s_7) pour solution initiale

On choisit s_7 aléatoirement parmi les successeurs de s_9 qu'est aussi choisi aléatoirement en S (figure 4.4), S n'est pas un ensemble dominant on continue.

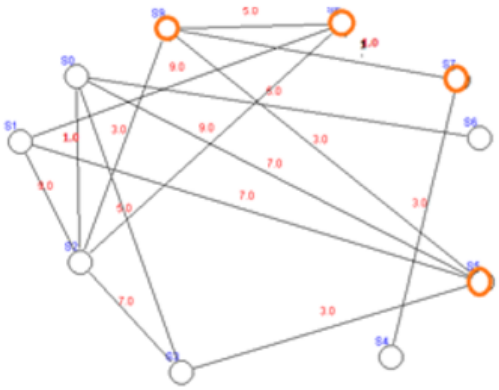

FIGURE 4.5 – choix aléatoire du 4ème sommet (s_5) pour solution initiale

Supposant que le sommet choisi dans S est aussi s_9 et le sommet choisi parmi ces succes-

seurs est s_5 (figure 4.5) alors $S = [\{s_9, s_8, s_7, s_5\}]$, S n'est pas un ensemble dominant on continue.

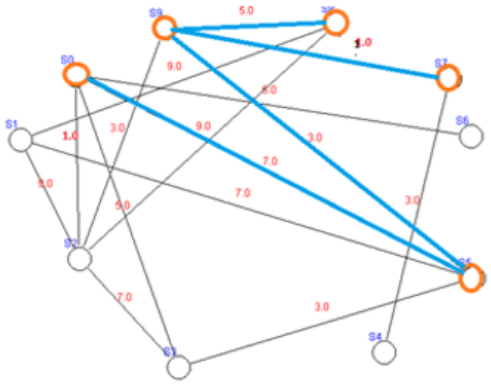

FIGURE 4.6 – choix aléatoire du 5ème sommet (s_0) pour solution initiale

Dans cette étape, le sommet choisi dans S est s_5 et le sommet choisi parmi ces successeurs est s_0 (figure 4.6) alors $S = [\{s_9, s_8, s_7, s_5, s_0\}]$,Cet ensemble est dominant alors on peut construire une solution qu'est :

Les sommets de la solution

$\{s_9, s_8, s_7, s_5, s_0\}$

Le reste des sommets de graphe

$\{s_1, s_2, s_3, s_4, s_6\}$

Le cout total est 12

4.2.4 Evaluation

Après la génération de population initiale, il faut calculer le coût de chaque solution. La fonction objectif dans notre problème est la somme des coûts des arêtes construisant l'arbre dominant.L'algorithme d'évaluation est noté (Algorithme 4.2)

Algorithm 4.2 Evaluation d'une solution

1: S : une solution
2: $Cout = 0$
3: **pour** $i = 1$ jusqu'à $taille(s.v\_cout)$ **faire**
4: $\quad Cout = Cout + s.v\_cout[i]$
5: **fin pour**
6: **retourner** $Cout$

Génération du voisinage

La génération du voisinage est une étape importante dans le BSO. En effet, si les voisins sont très différents de la solution courante, le BSO risque de se perdre. En revanche s'il y

a une grande similitude entre la solution courante et son voisin, l'intensification dominera la diversification, et cela augmenterait le risque de stagnation dans un minimum local. Afin d'assurer l'équilibre dans les l'algorithme nous proposons une solution à ce problème. Cette solution est d'utiliser deux types de voisinage :

Le voisinage_1 : consiste de générer une solution voisine d'une façon aléatoire comme suit :

On choisit un sommet s_1 aléatoire de la solution, et on l'enlève de l'arbre dominant de cette solution. On choisit aléatoirement un autre sommet s_2 et on le permute avec le premier sommet dans la solution (arbre dominant). L'algorithme de voisinage_1 est noté (Algorithme 4.3).

Algorithm 4.3 Algorithme voisinage_1

1: S : une solution
2: $r_1 \longleftarrow random(0, taille(S))$
3: Supprimer $(S[r_1])$
4: $r_2 \longleftarrow random(0, taille(S))$
5: Permuter $(S[0], S[r])$
6: $S\_n \longleftarrow$ créer_ensemble_dominant(S)
7: $S\_v \longleftarrow$ créer_arbre_dominant (S_n)
8: **Retourner** S_v

Exemple :

Supposant que :

$S =$

0	5	4	12	8	1

$R_1 = 3$

$S =$

0	5	4	8	1

$R_2 = 2$

$S =$

4	5	0	8	1

Le voisinage_2 : consiste à prendre les deux sommets ayant l'arête du coût maximal et supprime l'un qui a le degré minimal. L'algorithme de voisinage_2 est noté (Algorithme 4.4).

Algorithm 4.4 Algorithme voisinage_2

1: S : solution
2: $A \longleftarrow max(s.cout)$
3: s_1, s_2 les sommets reliés par a
4: $s_3 \longleftarrow choix\_min\_degre(s_1, s_2)$
5: supprimer (s_3)
6: $s\_n \longleftarrow$ créer_ensemble_dominant(S)
7: $S\_v \longleftarrow$ créer_arber_dominant (S_n)
8: **Retourner** S_v

Exemple :

Supposant que :

S=[[0, 5, 4, 8],[(0, 5),(5, 4),(4, 8),(0, 4)],[0.8, 4.2, 3.2, 1.2]]

Max = 4.2 alors on prend les sommets 4 et 5

Le degré du sommet 4 = 3

Le degré du sommet 5 = 2

Alors on supprime le sommet n° 5

$S =$

0	4	8

- on a opté pour le voisinage_1, car dans notre cas il a donné des meilleurs résultats.

4.3 Conclusion

Dans ce chapitre, nous avons développé les principales phases de l'algorithme d'optimisation par essaim d'abeilles (BSO), afin de résoudre le problème DT. Dans le chapitre suivant, nous allons implémenter et expérimenter notre algorithme puis interpréter les résultats.

Chapitre 5

Résultats et expérimentation

Sommaire

Ce dernier chapitre est consacré à l'étude expérimentale de l'algorithme présenté dans les chapitres précédents (BSO). Dans ce chapitre, nous présentons notre environnement de travail, les données de test utilisées dans l'expérimentation de notre approche. Nous donnons ainsi une présentation détaillée de notre application développée. Ensuite, nous présentons les résultats obtenus lors de l'application de notre approche BSO sur les données de test. Nous analysons ces résultats en les comparant avec des résultats obtenus par la méthode VNS (variable neighborhood search) développée par [26] Zorica Dražić, Mirjana Čangalović et Vera Kovačević-Vujčić.

5.1 Environnement de travail

L'implémentation de l'algorithme a été effectué en langage java en utilisant l'environnement de programmation : Netbeans IDE 8.2, sous système d'exploitation Windows 10. Les expérimentations ont été menées sur un PC de type i5- 4200U CPU @ 1.6GHZ 2.30GHZ et de RAM 4Go

5.2 Données de test

Les instances de travail sont disponibles sur le site `http://poincare.matf.bg.ac.rs/~zdrazic/dtp`. Ces instances sont générées aléatoirement avec différents nombres des sommets et des arêtes. Le nombre des sommets est varié entre 10 et 300 et le nombre des arêtes est varié entre 15 et 1000 Il y a deux types des instances « dtp_small » et « dtp_large » selon la taille du graphe.
Les instances « dtp_small » contiennent des graphes ayant un nombre de sommets entre 10 et 20 .pour chaque graphe il y a trois modèles comme par exemple le graphe qui contient 10 sommets et 15 arêtes il y a « dtp_10_15_0 », « dtp_10_15_1 » et « dtp_10_15_2 ». Les instances « dtp_large » représentent des graphes ayant un nombre de sommets entre 100 et 300 avec nombre d'arêtes entre 150 et 1000 comme dans les instances « dtp_small », pour chaque graphe il y a trois modèles comme par exemple le graphe qui contient 10 sommets et 15 arêtes il y a « dtp_100_150_0 », « dtp_100_150_1 » et « dtp_100_150_2 ».
Chaque instance est représentée dans un fichier texte et organisée comme le montre la figure 5.1

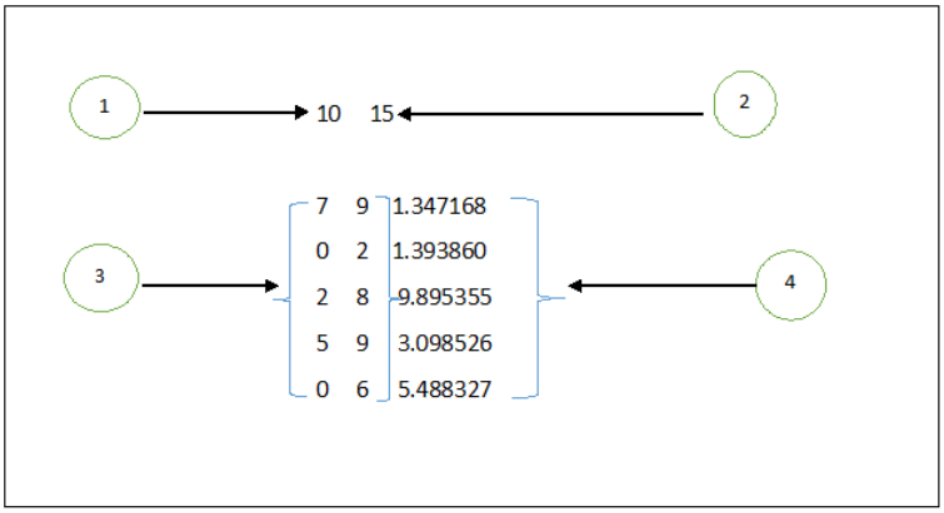

FIGURE 5.1 – Représentation d'une instance dans un fichier texte

1 : représente le nombre des sommets.

2 : représente le nombre des arêtes

3 : représente la matrice d'adjacence des sommets.

4 : représente le coûts entre chaque deux sommets.

5.3 Présentation de l'interface

Le développement de notre application s'articule autour d'une fenêtre principale, cette fenêtre est décrite en détail dans la section suivante. La fenêtre principale de l'application est illustrée par la figure 5.2, elle est composée de trois blocs :

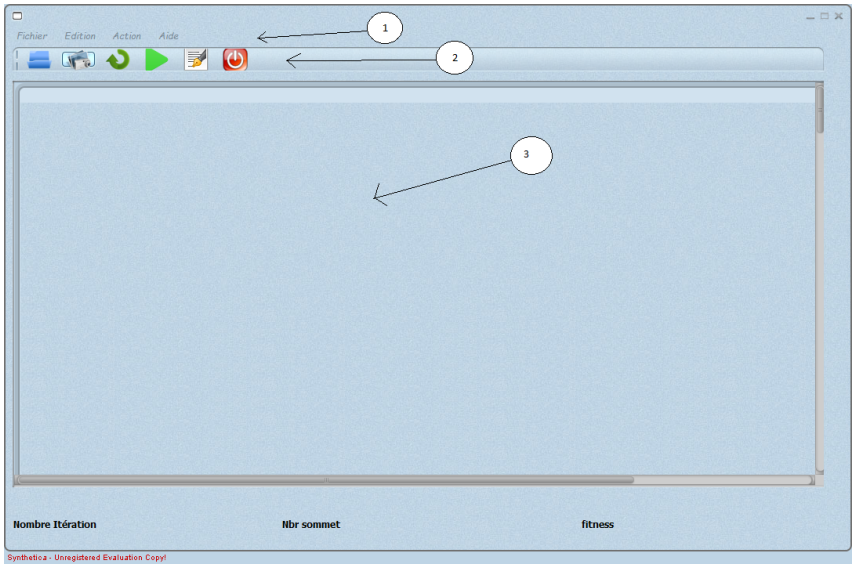

FIGURE 5.2 – Fenêtre principale de l'application

✓ **Bloc 1 :** Il s'agit d'une barre de menu qui regroupe toutes les fonctionnalités offertes par l'application, elle contient 4 menus comme le montre la figure 5.3

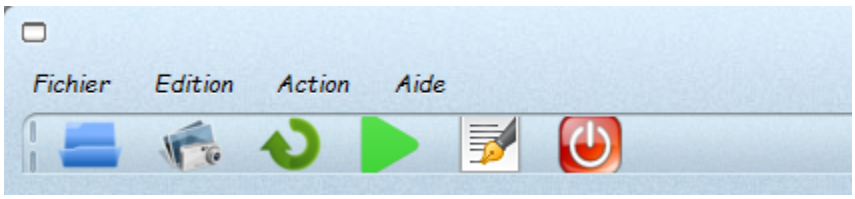

FIGURE 5.3 – barre de menu

➢ **Le menu Fichier :** Il propose deux fonctions :

1. Charger un fichier : En sélectionnant cette fonction, la fenêtre de la figure 5.5 s'affiche. Cette fenêtre contient 3 sous blocs : **le premier sous bloc** contient un bouton qui permet de déterminer le chemin d'accès au (aux) fichier(s) qui sont déjà stocké(s) en

mémoire (Disque dur) et qui contiennent les instances. Enfin, en cliquant sur Ouvrir, la sélection du fichier est validée comme illustré dans la figure 5.4.

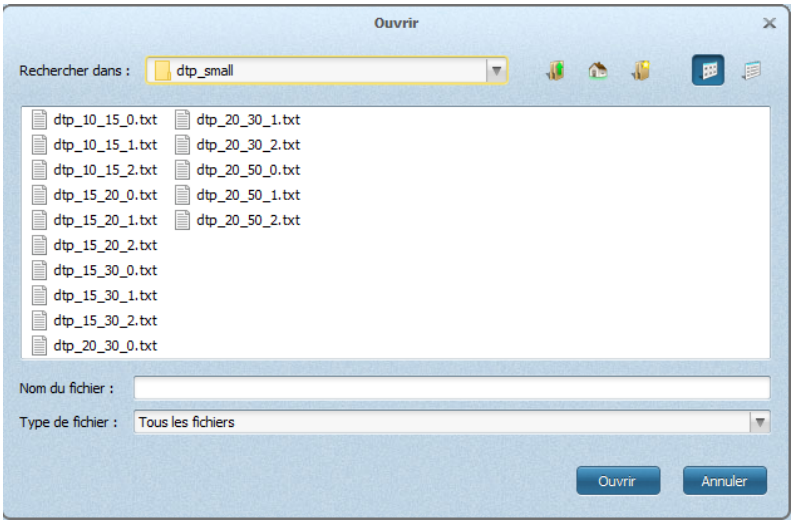

FIGURE 5.4 – Chemin d'accès au (aux) fichier(s) des instances

Le deuxième sous bloc ppermet de préciser les valeurs des paramètres de l'algorithme d'optimisation par essaims d'abeilles (BSO) .

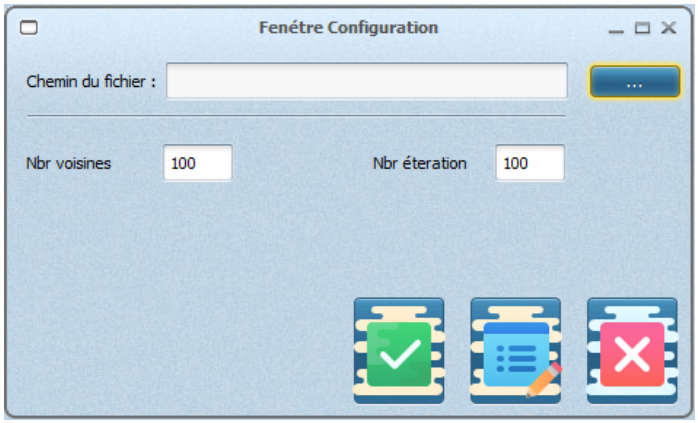

FIGURE 5.5 – Précision les valeurs des paramètres de l'algorithme BSO.

2. Quitter : Permet de fermer l'application.

➢ **Le menu Edition :** Ce menu contient un seul élément :

1. Run : Permet de démarrer l'exécution.

➢ **Le menu Action :** Contient les fonctions :

1. Capturer : Permet de générer une image de la fenêtre en cours.

2. Actualiser : actualiser l'affichage.

➢ **Le menu Aide :** En sélectionnant l'option Manuel, un guide facilitant l'utilisation de l'application peut être visualisé.

✓ **Bloc 2 :** Dans ce bloc se trouve la barre des raccourcis (barre d'outils) qui regroupe les raccourcis des fonctions principales de la barre des menues cités dans la figure 5.6.

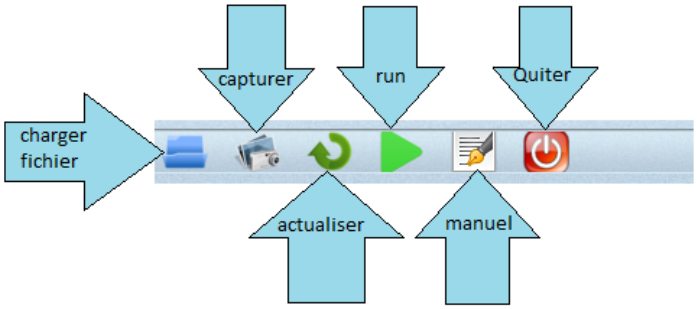

FIGURE 5.6 – la barre des raccourcis (barre d'outils) de l'application

✓ **Bloc 3 :** Ce bloc permet à l'utilisateur de visualiser le graphe et l'arbre dominant de se graphe comme le montre la figure 5.7

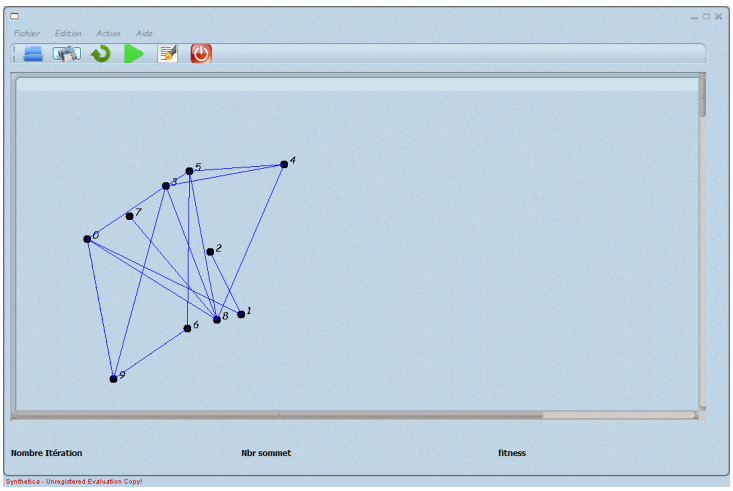

FIGURE 5.7 – fenêtre de visualisation du graphe et de l'arbre de dominant

5.4 Résultats et comparaison

Cette section est consacrée à l'évaluation des performances de l'algorithme BSO ainsi que la comparaison avec les résultats obtenus par VNS et les solutions optimales pour les petites instances.

Nous comparons les résultats obtenus pour le jeu de données utilisé à l'aide de l'opérateur GAP qui calcule l'écart entre deux solutions, et qui est défini dans l'équation 5.1 :

$$GAP_{Approche1-Approche2} = \frac{Cout_{Approche1} - Cout_{Approche2}}{Cout_{Approche2}} \times 100\% \qquad (5.1)$$

5.4.1 Comparaison avec les petites instances

Les solutions pour les petites instances sont optimales, le tableau 5.1 représente les résultats obtenus par l'approche VNS et les solutions optimale. La première colonne représente le nom de l'instance, la deuxième colonne contient les résultats optimales en utilisant la méthode exacte CPLEX, la troisième colonne représente les résultats obtenus par la méthode VNS [26]. La dernière colonne contient le nombre des arêtes dans la solution trouvée par VNS.

Instance	CPLEX	VNS	ANDV
dtp_10_15_0	5.89	5.89	4
dtp_10_15_1	14.42	14.42	5
dtp_10_15_2	14.35	14.35	4
dtp_15_20_0	18.87	18.87	6
dtp_15_20_1	23.03	23.03	6
dtp_15_20_2	24.95	24.95	6
dtp_15_30_0	18.20	18.20	5
dtp_15_30_1	8.32	8.32	4
dtp_15_30_2	18.07	18.07	6
dtp_20_30_0	33.81	33.81	9
dtp_20_30_1	36.03	36.03	8
dtp_20_30_2	43.50	43.50	10
dtp_20_50_0	9.81	9.81	5
dtp_20_50_1	-	12.19	6
dtp_20_50_2	17.42	17.42	6

TABLE 5.1 – Résultats obtenus par l'approche VNS et les solutions optimale(CPLEX) pour les petites instances

Tous les résultats obtenu par VNS sont optimales. le tableau 5.2 représente les résultats obtenu par notre approche proposée BSO.

Instance	BSO	ANDV
dtp_10_15_0	5.89	4
dtp_10_15_1	14.42	5
dtp_10_15_2	14.35	4
dtp_15_20_0	18.87	6
dtp_15_20_1	23.03	6
dtp_15_20_2	24.95	6
dtp_15_30_0	18.20	5
dtp_15_30_1	8.32	4
dtp_15_30_2	18.07	6
dtp_20_30_0	33.81	9
dtp_20_30_1	36.03	8
dtp_20_30_2	43.50	10
dtp_20_50_0	9.81	5
dtp_20_50_1	12.19	6
dtp_20_50_2	17.42	6

TABLE 5.2 – Résultats obtenus par notre approche BSO pour les petites instances

Nous remarquons que sur la plupart des instances, les résultats obtenus par notre approche sont optimales. Cela est due au fait que l'espace de recherche est réduit.
Le tableau 5.3 récapitule les résultats de comparaisons entre BSO et VNS pour les petites instances

✓ **BSO vs VNS (pour les petites instances)**

Instance	VNS		BSO		différence	
	sol	ANDV	sol	ANDV	Δ(différence)	GAP(%)
dtp_10_15_0	5.89	4	5.89	4	0	0
dtp_10_15_1	14.42	5	14.42	5	0	0
dtp_10_15_2	14.35	4	14.35	4	0	0
dtp_15_20_0	18.87	6	18.87	6	0	0
dtp_15_20_1	23.03	6	23.03	6	0	0
dtp_15_20_2	24.95	6	24.95	6	0	0
dtp_15_30_0	18.20	5	18.20	5	0	0
dtp_15_30_1	8.32	4	8.32	4	0	0
dtp_15_30_2	18.07	6	18.07	6	0	0
dtp_20_30_0	33.81	9	33.81	9	0	0
dtp_20_30_1	36.03	8	36.03	8	0	0
dtp_20_30_2	43.50	10	43.50	10	0	0
dtp_20_50_0	9.81	5	9.81	5	0	0
dtp_20_50_1	12.19	6	12.19	6	0	0
dtp_20_50_2	17.42	6	17.42	6	0	0

TABLE 5.3 – Résultats BSO Vs VNS pour les petites instances

L'approche BSO converge vers la solution optimale pour la plupart des instances ainsi que l'approche VNS.

la Représentation graphique des résultats (BSO Vs VNS) est dans la figure 5.8

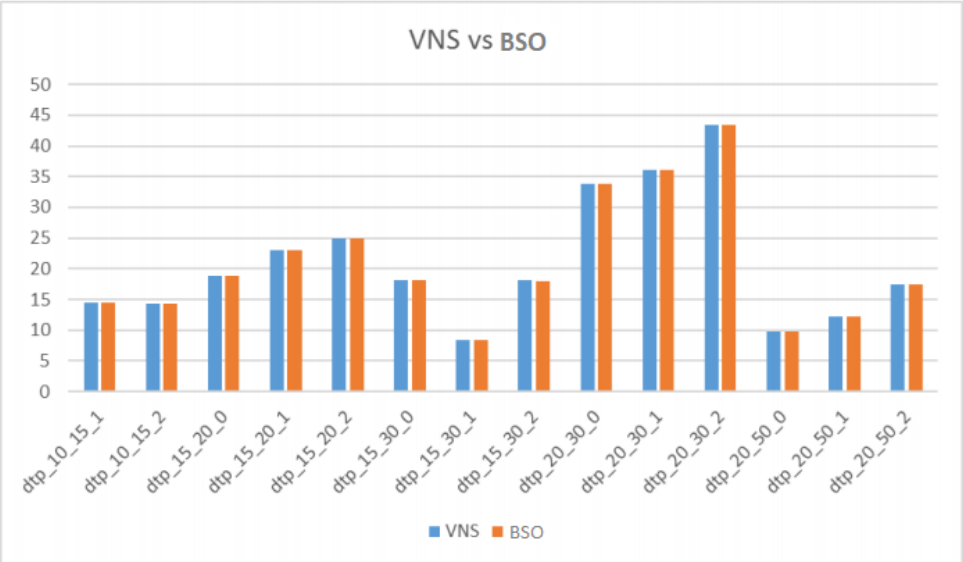

FIGURE 5.8 – Représentation graphique des résultats (BSO Vs VNS) pour les petites instances

Pour les petites instances, les deux approches ont donné des résultats optimaux. L'algorithme BSO est stable pour les petites instances car il atteint toujours le résultat optimal pour chaque expérimentation et avec un petit nombre d'itérations.

5.4.2 Comparaison avec les grandes instances

Le tableau 5.4 représente les résultats obtenus par l'approche VNS.La première colonne représente le nom de l'instance, la deuxième colonne représente la meilleure solution trouvée par VNS pour chaque instance.

Instance	VNS (sol_best)
dtp_100_150_0	152.57
dtp_100_150_1	192.21
dtp_100_150_2	146.34
dtp_100_200_0	135.04
dtp_100_200_1	91.88
dtp_100_200_2	115.93
dtp_200_400_0	306.06
dtp_200_400_1	303.53
dtp_200_400_2	274.37
dtp_200_600_0	132.49
dtp_200_600_1	162.92
dtp_200_600_2	139.08
dtp_300_600_0	471.69
dtp_300_600_1	494.91
dtp_300_600_2	500.72
dtp_300_1000_0	257.72
dtp_300_1000_1	242.79
dtp_300_1000_2	223.18

TABLE 5.4 – Résultats obtenus par l'approche VNS pour les grandes instances

Le tableau 5.5 représente les résultats obtenu par l'approche BSO. La première colonne représente le nom de l'instance, la deuxième colonne représente la meilleure solution trouvée par notre approche BSO pour chaque instance.

Instance	BSO
dtp_100_150_0	152.57
dtp_100_150_1	192.21
dtp_100_150_2	149.74
dtp_100_200_0	135.2
dtp_100_200_1	91.88
dtp_100_200_2	116.9
dtp_200_400_0	277.38
dtp_200_400_1	286.83
dtp_200_400_2	267.24
dtp_200_600_0	134.74
dtp_200_600_1	151.51
dtp_200_600_2	142.56
dtp_300_600_0	392.37
dtp_300_600_2	481.63
dtp_300_1000_0	169.9
dtp_300_1000_1	218.55
dtp_300_1000_2	203.24

TABLE 5.5 – Résultats obtenus par notre approche BSO pour les grandes instances

✓ BSO vs VNS (pour les grandes instances)

Dans le tableau 5.6, nous présentons les résultats de comparaison des deux approches VNS et BSO. La première colonne représente le nom de l'instance, la deuxième colonne représente la meilleure solution trouvée par VNS pour chaque instance. la troisième colonne représente la meilleure solution trouvée par notre approche pour chaque instance. La dernière colonne contient la différence entre les résultats des deux approches en utilisant les deux opérateurs Δ et GAP tel que :

$\Delta = RES\_VNS - RES\_BSO$

$GAP(\%) = \frac{RES\_VNS - RES\_BSO}{RES\_BSO} \times 100\%$

Instance	VNS	BSO	différence	
	sol	sol	Δ(différence)	GAP(%)
dtp_100_150_0	152.57	152.57	00	0
dtp_100_150_1	192.21	192.21	00	0
dtp_100_150_2	146.34	149.74	-3.4	-2.27
Moyenne	**163.7**	**164.84**	**-1.14**	**-0.69**
dtp_100_200_0	135.04	135.2	-0.16	-0.12
dtp_100_200_1	91.88	91.88	00	00
dtp_100_200_2	115.93	116.9	-0.97	-0.83
Moyenne	**114.28**	**114.66**	**-0.38**	**-0.33**
dtp_200_400_0	306.06	277.38	28.68	10.34
dtp_200_400_1	303.53	286.83	16.7	5.82
dtp_200_400_2	274.37	267.24	7.13	2.67
Moyenne	**294.65**	**277.15**	**17.5**	**6.31**
dtp_200_600_0	132.49	134.74	-2.25	-1.67
dtp_200_600_1	162.92	151.51	11.41	7.53
dtp_200_600_2	139.08	142.56	-3.48	-2.44
Moyenne	**144.83**	**142.93**	**1.9**	**1.32**
dtp_300_600_0	471.69	392.37	79.32	20.21
dtp_300_600_2	500.72	481.63	19.09	3.96
Moyenne	**486.2**	**437**	**49.2**	**11.26**
dtp_300_1000_0	257.72	169.9	87.82	51.68
dtp_300_1000_1	242.79	218.55	24.24	11.09
dtp_300_1000_2	223.18	203.24	19.94	9.81
Moyenne	**241.23**	**197.23**	**44**	**22.3**
Moyenne générale	**240.81**	**222.3**	**18.68**	**8.4**

TABLE 5.6 – Résultats BSO Vs VNS pour les grandes instances

La représentation graphique des résultats BSO Vs VNS pour les grandes instances est dans la figure 5.9

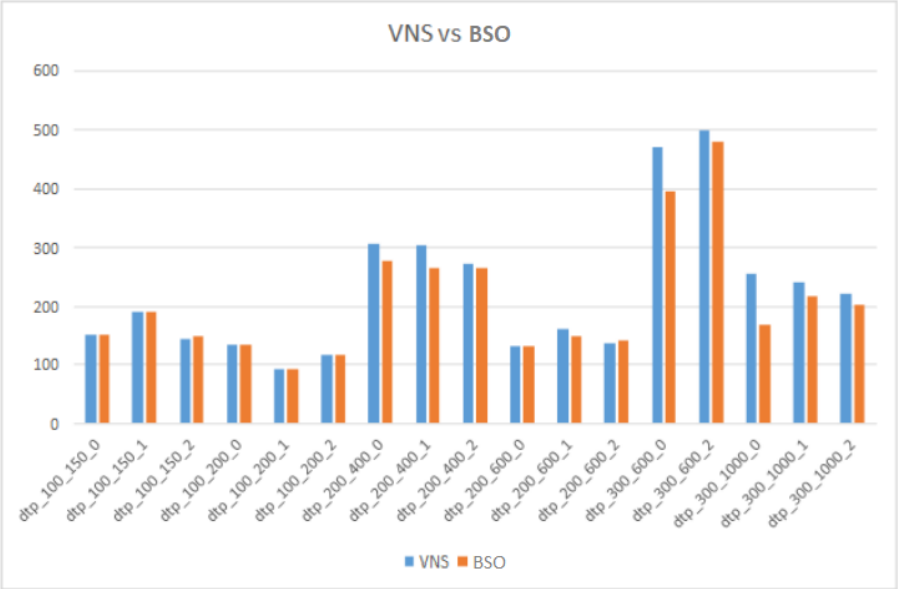

FIGURE 5.9 – Représentation graphique des résultats (BSO Vs VNS) pour les grandes instances

D'après la figure 5.9 et le tableau 5.6 nous remarquons que pour les instances de 100 sommets et 150 arêtes les résultats obtenues par BSO sont soit très proches aux résultats de VNS soit sont égaux.

Pour les instances de 100 sommets et 200 arrêtes, les résultats obtenus par BSO sont très proches aux résultats de VNS.

Pour les instances de 200 sommets et 400 arêtes, notre approche donne des solutions meilleures que ceux de VNS. L'approche proposée a amélioré VNS dans ces instances avec une moyenne de de 6.31 %.

Pour les instances de 200 sommets et 600 arêtes, notre algorithme BSO donne des meilleurs résultats que l'approche VNS et améliore cette dernière avec un pourcentage de 1.32%.

Pour les instances de 300 sommets et 600 arêtes les résultats de notre approche dépassent les résultats de VNS par une moyenne de 11.26%.

Pour les instances de 300 sommets et 1000 arêtes, les résultats de notre approche sont meilleurs que ceux de VNS. BSO a amélioré VNS par une moyenne de 22.3%

D'après la moyenne générale des résultats, nous concluons que notre algorithme développé BSO est plus performant que VNS pour ces instances.

5.5 Conclusion

Dans ce dernier chapitre, nous avons présenté les résultats obtenus par l'algorithme proposé dans le chapitre précédent à savoir l'algorithme d'optimisation par essaims d'abeilles BSO pour résoudre le problème de l'arbre dominant DT. L'étude expérimentale a montré l'efficacité de l'approche proposée BSO pour les petites et les grandes instances. Nous concluons que BSO représente une bonne approche pour résoudre le problème de l'arbre dominant.

Conclusion générale et perspectives

Le travail élaboré dans ce mémoire porte essentiellement sur le problème de l'arbre dominant DTP, qui permet la modélisation dans le domaine des réseaux de capteurs sans fil (WSN).

L'objectif était d'appliquer l'algorithme d'optimisation par essaims pour résoudre le problème cité. Les trois premiers chapitres étaient consacrés aux études théoriques. Dans le premier chapitre, nous avons présentés des notions liées à l'optimisation combinatoire. Ensuite, nous avons introduit la notion des métaheuristiques en présentant des généralités sur ces derniers. Deux types de méthodes de résolution des problèmes NP-Difficile tel que le problème de l'arbre dominant DTP, ont été abordées dans ce chapitre : les méthodes exactes qui assurent l'optimalité de la solution mais pour des problèmes de petites instances (taille limitée), et les approches heuristiques et métaheuristiques qui fournissent des solutions appréciables en un temps raisonnable a des problèmes de n'importe quelle instance.

Dans le chapitre 2 nous avons présenté en détails les étapes de l'algorithme d'optimisation par essaims d'abeilles pour l'optimisation des problèmes continus.Dans le chapitre 3, nous avons présenté quelques notions de bases sur les graphes. Ensuite, nous avons présenté le problème de l'arbre dominant et les problèmes connexes au problème DT.

Ensuite, nous avons proposé notre approche de résolution du problème de l'arbre dominant DTP, cette approche était basée principalement sur l'algorithme BSO .

Nous avons testé notre approche sur différents ensembles générés de manière aléatoire du problème de l'arbre dominant. Ces ensembles contiennent 33 instances de DTP, nous avons testé notre approche sur 32 instances (3 instances de 10 sommets et 15 arêtes, 3 instances de 15 sommets et 20 arêtes, 3 instances de 15 sommets et 30 arêtes, 3 instances de 20 sommets et 30 arêtes, 3 instances de 20 sommets et 50 arêtes, 3 instances de 100 sommets et 150 arêtes, 3 instances de 100 sommets et 200 arêtes, 3 instances de 200 sommets et 400 arêtes, 3 instances de 200 sommets et 600 arêtes, 2 instances de 300 sommets et 600 arêtes, 3 instances de 300 sommets et 1000 arêtes). Les résultats sont comparés avec les résultats optimaux pour les instances de petites tailles (10 et 20 sommets), pour les instances de grande taille (100, 200 et 300 sommets) nous avons comparés nos résultats

avec les meilleurs résultats trouvés dans la littérature. Les tests effectués sur ces instances ont montré que notre approche donne des résultats optimaux pour les instances de petite taille (10 et 20 sommets).
Pour la plupart des instances de grande taille (100, 200 et 300 sommets), nous arrivons à des solutions plus performantes que celles des VNS et pour les instances qui restent nous arrivons à des solutions très proches.
Les résultats obtenus dans l'expérimentation montrent l'efficacité de l'approche proposée.

Bibliographie

[1] Marisa Maximiano da Silva. *Aplicando metaheurística multiobjetivo al problema de asignación de frecuencias en redes GSM : Applying multiobjective metaheuristics to the frequency assignment problem in GSM networks.* PhD thesis, Universidad de Extremadura, 2011.

[2] Jin-Kao Hao, Philippe Galinier, and Michel Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, 13(2) :283–324, 1999.

[3] Mohamed Ekbal Bouzgarrou. *Parallélisation de la méthode du" Branch and Cut" pour résoudre le problème du voyageur de commerce.* PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1998.

[4] ABDESSELAM Salim. *Conception d'un système d'optimisation pour le positionnement de caméras pour la motion capture MOCAP (Utilisation des métaheuristiques OEP et RFS).* PhD thesis, Université Mohamed Khider-Biskra, 2018.

[5] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[6] Nelson. Maculan and C. C. Ribeiro. *Applications of combinatorial optimization / editors : Celso Carneiro Ribeiro, Nelson Maculan.* J.C. Baltzer Basel, Switzerland, 1994.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[8] H Jin-Kao, P Galinier, and M Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2) :283–324, 2000.

[9] Y C Ho and D L Pepyne. Simple Explanation of the No-Free-Lunch Theorem and Its Implications. *Journal of Optimization Theory and Applications*, 115(3) :549–570, 2002.

[10] Eric Bonabeau, Directeur de Recherches Du Fnrs Marco, Marco Dorigo, Guy Theraulaz, et al. *Swarm intelligence : from natural to artificial systems*. Number 1. Oxford university press, 1999.

[11] MJ Couvillon. The dance legacy of karl von frisch. *Insectes sociaux*, 59(3) :297–306, 2012.

[12] Thomas D Seeley, Scott Camazine, and James Sneyd. Collective decision-making in honey bees : how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, 28(4) :277–290, 1991.

[13] Theraulaz Bonabeau. *intelligence collective*. PhD thesis.

[14] Habiba Drias, Souhila Sadeg, and Safa Yahi. Cooperative bees swarm for solving the maximum weighted satisfiability problem. In *International Work-Conference on Artificial Neural Networks*, pages 318–325. Springer, 2005.

[15] Hélène Topart. *Etude d'une nouvelle classe de graphes : les graphes hypotriangulés*. PhD thesis, Conservatoire national des arts et metiers-CNAM, 2011.

[16] Reinhard Diestel. *Graphe theory*. Springer, 2000.

[17] Nasser Eddine MOUHOUB. *Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet*. PhD thesis, Université Farhat Abbas de Sétif.

[18] Mohammed Tamali. Théorie des graphes. 2013.

[19] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4) :374–387, 1998.

[20] Jeanne Gaillard. Paris la ville, 1852-1870. *Le Mouvement social*, pages 3–13, 1975.

[21] Colorations des sommets ; wwwpolymtl.ca/pub/site/la grapheur/docs/documents/notechap7.pdf.

[22] Incheol Shin, Yilin Shen, and My T Thai. On approximation of dominating tree in wireless sensor networks. *Optimization Letters*, 4(3) :393–403, 2010.

[23] Shyam Sundar and Alok Singh. New heuristic approaches for the dominating tree problem. *Applied Soft Computing*, 13(12) :4695–4703, 2013.

[24] My T Thai, Feng Wang, Dan Liu, Shiwei Zhu, and Ding-Zhu Du. Connected dominating sets in wireless networks with different transmission ranges. *IEEE transactions on mobile computing*, 6(7) :721–730, 2007.

[25] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1597–1604. IEEE, 2002.

[26] Zorica Dražić, Mirjana Čangalović, and Vera Kovačević-Vujčić. A metaheuristic approach to the dominating tree problem. *Optimization Letters*, pages 1–13, 2016.

[27] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6) :1087–1092, 1953.

[28] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4) :374–387, 1998.

[29] Myung Ah Park, James Willson, Chen Wang, Weili Wu, and Andras Farago. A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 22–31. ACM, 2007.

[30] My T Thai, Feng Wang, Dan Liu, Shiwei Zhu, and Ding-Zhu Du. Connected dominating sets in wireless networks with different transmission ranges. *IEEE transactions on mobile computing*, 6(7), 2007.

[31] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *INFOCOM 2002. Twenty-First annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE*, volume 3, pages 1597–1604. IEEE, 2002.

Résumé

Le problème de l'arbre dominant (DTP) consiste à trouver un arbre, disons DT, avec un poids minimum total des arêtes sur un graphe non orienté, pondéré et connexe, de sorte que chaque sommet du graphe soit en DT ou adjacent à un sommet en DT.

Dans ce mémoire, nous présentons une nouvelle approche d'optimisation par essaims d'abeilles (BSO) pour la résolution de ce problème.

Les résultats obtenus dans l'expérimentation montrent l'efficacité de l'approche proposée.

Mots clés : DTP, Problème d'arbre dominant, Arbre dominant, Graphes, algorithme d'optimisation par essaims d'abeilles , Heuristique , Optimisation , réseaux de capteurs sans fil.

Abstract

The dominating tree problem (DTP) consists in finding a tree, say DT, with minimum total edge weight on an undirected, weighted and connected graph such that each vertex of the graph is either in DT or adjacent to a vertex in DT.

In this paper, we present a new approach of improved bee swarm optimization (BSO) to solve this problem.

The results obtained in the experiment show the effectiveness of the proposed approach.

Keywords : DTP, Dominating tree problem, Dominating tree, Graphs, Bee swarm optimization algorithm, Heuristic, optimization, Wireless Sensor Networks.