Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Mémoire de Master

Domaine : Informatique

Spécialité : Systèmes Informatiques Intelligents

Thème

MINIMISATION DE LA CONSOMATION D'ENERGIE DANS UN RESAUX DE CAPTEURS (WSN) PAR UNE APPROCHE COOPERATIVE DE METHAHEURISTIQUE

Présenté par :
- EL DJAZAERY IBRAHIM
- CHEKLAT KHADIDJA

Proposé et dirigé par :
- Pr. BOUKRA ABDELMADJID
- Mme. OUAFI MOUNIRA

Projet N° : 085/2019

Remerciment

Nous tenons à remercier en tout premier lieu ALLAH tout puissant de nous avoir donné la volonté et la puissance d'élaborer ce modeste travail.

Nous adressons nos vifs remerciements à nos encadreurs Pr. BOUKRA et Mme OUAFI qui nous a formulé ses précieux conseils et qui nous a facilité la tâche par ses recommandations et ses orientations.

Nos remerciements vont également à toute personne ayant apporté son aide de près ou de loin, à la réalisation de ce travail. Notamment nos amis et nos familles respectives qui nous ont fourni l'environnement adéquat pour la mise en œuvre de notre projet.

Nous remercions les membres du jury pour avoir accepté d'examiner et de juger ce modeste travail.

Enfin nous souhaitons adresser nos remerciements au corps enseignant du département informatique, Faculté d'Electronique et d'Informatique –USTHB-, pour l'enseignement de qualité et qui par leurs conseils et leurs critiques ont guidé nos réflexions.

Table des matières

Table des figures

Liste des Algorithmes

Introduction Général

Les réseaux de capteurs sans fil sont récemment devenus un sujet de recherche de premier plan. L'évolution technologique de ces derniers a permis une production de masse peu coûteuse de nœuds de capteurs autonomes, qui, malgré leurs miniaturisation, ont des capacités de détection, de traitement et de communication particulièrement avancées [4] , et ont un fort potentiel économique à long terme. Flexibilité, hautes capacités de captage, coût réduit, installation rapide sont les caractéristiques qui ont permis aux réseaux de capteurs d'avoir des nouveaux domaines d'applications multiples et excitants. Ce large étendu d'application fera de cette technologie émergente une partie intégrale de nos vies futures.

Cependant, la réalisation des réseaux de capteurs nécessite la satisfaction de certaines contraintes qui découlent d'un nombre de facteurs guidant la phase de conception, tel que la tolérance aux pannes, la scalabilité, le coût et la consommation de l'énergie qui est la substance de notre sujet. Le problème majeur des réseaux de capteurs sans fil est le problème des trous d'énergie. Les nœuds les plus proches de la région de puits mourront plus tôt des sous-régions externes car ils envoient leurs propres données et transmettent également les données des sous-régions externes au récepteur. Donc, après très peu de temps, un trou d'énergie vient près de la région de l'évier. Après cela, les données ne peuvent plus être transmises, même s'il reste encore de l'énergie dans les nœuds de la région externe, ce qui affecte la durée de vie du réseau. Par conséquent, pour permettre d'augmenter la longévité, d'un nœud capteur nous nous intéressons particulièrement à la résolution d'un problème d'optimisation combinatoire, qui est relativement nouveau dans les RCSFs, intitulé le problème de l'arbre dominant ou DTP. Ce dernier offre une ossature pour le routage des données à travers le réseau, du fait que les informations de routage seront stockés uniquement au niveau des nœuds de l'arbre dominant.

La résolution de ce problème qui est NP-Difficile [5, 6] , fait recours aux méthodes d'optimisation combinatoire, à savoir, les méta-heuristiques, tout en tirant profit des propriétés de la théorie des graphes. Ensuite, des essais d'amélioration sont apportés en travaillant sur ses paramètres afin d'obtenir les méthodes les plus efficaces possibles. La qualité de chaque méthode est évaluée en la comparant à d'autres méthodes proposées pour le problème étudié. Malheureusement, d'après les No Free Lunch Theorems

[Wolpert 1997], il n'existe pas de métaheuristique qui soit meilleure que toutes les autres métaheuristiques pour tous les problèmes. Dans la pratique, il existera toujours des instances pour lesquelles une métaheuristique est meilleure qu'une autre. Quelque soit la métaheuristique choisie, elle présente des avantages et des inconvénients. C'est la raison pour laquelle la méthode de collaboration a été proposée. Cette méthode propose de regrouper un ensemble d'heuristiques ou métaheuristiques et d'établir un mécanisme pour identifier et sélectionner les méthodes de recherche les plus efficaces au cours du processus d'optimisation. Généralement, certaines études visent à produire des heuristiques constructives qui construiront une solution, étape par étape. Les heuristiques sont utilisées pour décider comment étendre une solution partielle. Ces méthodes ont tendance à être rapide. D'autres études visent à produire des heuristiques d'amélioration ou de perturbation travaillant sur une solution candidate déterminée et essayant d'améliorer sa qualité. Ces méthodes sont plus lentes mais fournissent les meilleurs résultats finaux.

Notre mémoire s'articule autour de cinq chapitres. Le premier présente un état de l'art des Réseaux de Capteurs Sans Fils (RCSF). L'étude des méthodes d'optimisation combinatoires et de résolution exactes et approchées fait l'objet du chapitre deux. Le troisième chapitre sera consacré à l'étude du problème de l'arbre dominant ainsi que des différents travaux effectués pour sa résolution et sur lesquels repose notre travail. Dans le quatrième chapitre, nous présentons l'approche de résolution développée. Quant au chapitre cinq, il est consacré à l'expérimentation ainsi qu'à l'analyse des performances. Nous terminons par une conclusion générale et des perspectives.

Nous terminons par une conclusion générale et des perspectives.

premier chapter

1.1 Introduction

L'une des récentes avancées dans la micro électronique et des technologies sans-fil qui confortent la présence de l'informatique et de l'électronique au cœur du monde réel ont permis de développer des capteurs de petite taille. Ces micro-capteurs sont dotés d'une capacités de traitement permettant de collecter et de transmettre des données environnementales d'une manière autonome.Les réseaux de capteurs sans fil ou WSNs(Wireless Sensor Networks) sont constitués d'un ensemble de capteurs sans fil s'auto organisant pour acquérir des données de leur environnement immédiat, de les traiter et de les communiquer. Dans ce premier chapitre, nous présenterons un ensemble de généralités sur les réseaux de capteurs, notamment sur leur architecture et leurs domaines d'applications.

1.2 Un réseau de capteur sans-fil

Un réseau de capteur sans fil est un type spécial de réseau ad-hoc où la plupart de ces nœuds sont des micro-capteurs dispersés dans une zone géographique appelée zone de captage. La position de ces nœuds n'est pas obligatoirement déterminée, ils utilisent une communication sans fil pour acheminer les données captées avec un routage multi sauts vers un nœud collecteur appelé nœud puits(Sink). Ces capteurs comme ils ont été décrit au paravent sont des diapositifs de taille extrêmement réduite avec des sources limitées, autonome, capable de traiter et transmettre les informations.

Fig. 1.1 – Capteur sans fils

Les réseaux de capteurs utilisent un très grand nombre de ces capteurs, pour former un réseau sans infrastructure établie. Chaque capteur relayant l'information sur sa propre zone de couverture, le réseau se trouve entièrement couvert.

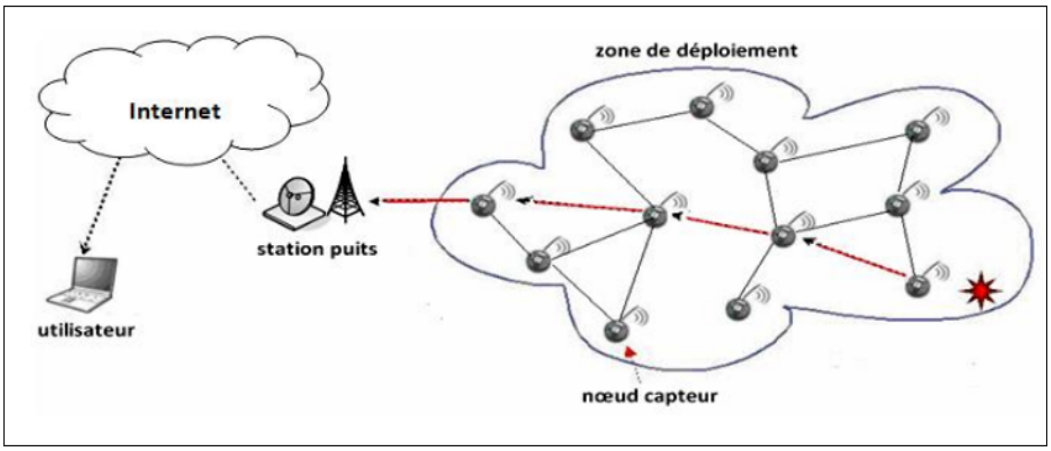

Fig. 1.2 – Architecture d'un réseau de capteurs sans fil

1.3 Architecture d'un capteur sans-fil

Un capteur est composé de quatre composants de base : Unité d'acquisition peut être nommé aussi unité de capture, unité de traitement, unité de communication(unité d'émission/réception), et une unité d'énergie. Il se peut aussi qu'il existe d'autres composants additionnels dépendant de l'application, par exemples : un générateur d'énergie, un système de localisation, et un mobilisateur.[7]

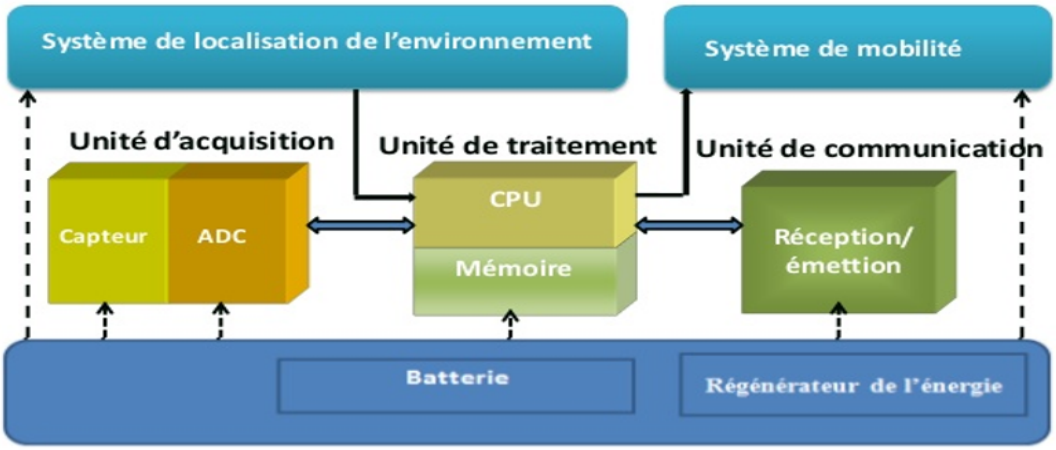

Fig. 1.3 – Architecture d'un nœud capteur

- **Unité d'acquisition :** Se compose généralement de deux sous unités : un capteur et un convertisseur(ADC) (analogique,numérique). Les signaux analogiques mesurés par le capteur sont convertis en signaux numériques (digitaux) et sont transmis à l'unité de traitement [7].

- **Unité de traitement :** L'unité de traitement, comprend un processeur associé généralement à une petite unité de stockage et fonctionne à l'aide d'un système d'exploitation spécialement conçu pour les micro-capteurs (TinyOS [8], par exemple). Cette unité est chargée d'exécuter les protocoles de communication qui permettent la collaboration entre les capteurs du réseau. Elle peut aussi analyser les données captées pour alléger la tâche des stations puits. De plus, l'unité de traitement nécessite un stockage pour minimiser la taille des messages transmis et cela en appliquant un traitement local et une agrégation de données [9].

- **Unité de communication :** Cette unité est responsable de l'émission et de la réception au moyen de support sans fil. Il existe trois schéma de communication pour les réseaux de capteurs : La communication optique (laser), L'infrarouge et la communication par fréquence radio. D'après les recherches scientifique la communication optique consomme moins d'énergie que la communication qui utilise la fréquence radio, son inconvénient est l'utilisation d'une ligne optique qui est sensible à la perturbation physique. L'infrarouge a une capacité de diffusion très limitée et il n'a pas besoin de l'antenne contrairement au

fréquence radio qui utilise des antennes mais très simple à l'utiliser.

- **Unité de contrôle d'énergie(batterie) :** C'est l'unité la plus importante. Son rôle est de distribuer de manière équitable l'énergie disponible entre les différentes unités et de réduire les dépenses en mettant en veille les composants inactifs

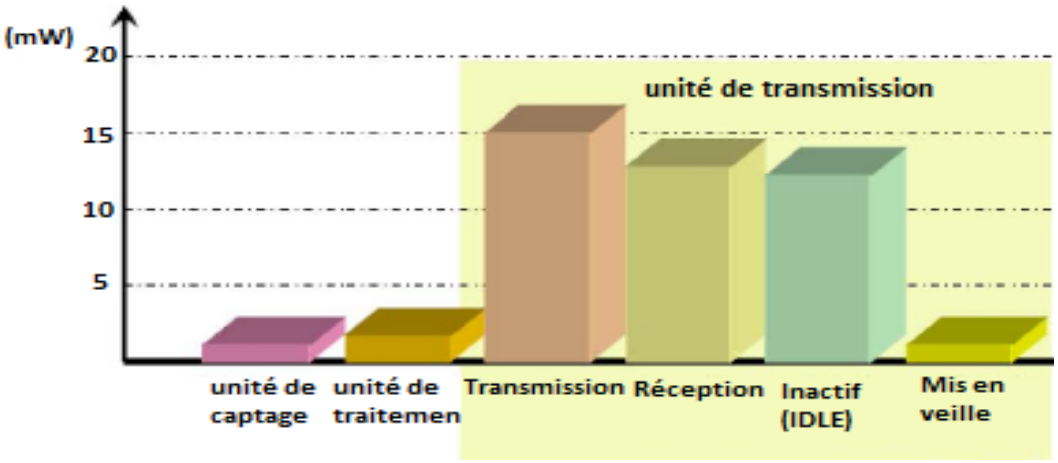

Fig. 1.4 – Consommation d'énergie dans un nœud capteur. [1, 2]

1.4 Architecture d'un réseau de capteurs sans-fil

La plupart des architectures communes à un WSN suivent le modèle OSI. En principe, dans le réseau de capteurs, nous avons besoin de cinq couches : couche application, couche de transport, couche réseau, couche liaison de données et couche physique. Les cinq couches sont complétées par les plans des trois couches, comme illustré à la Figure. 1.5 .

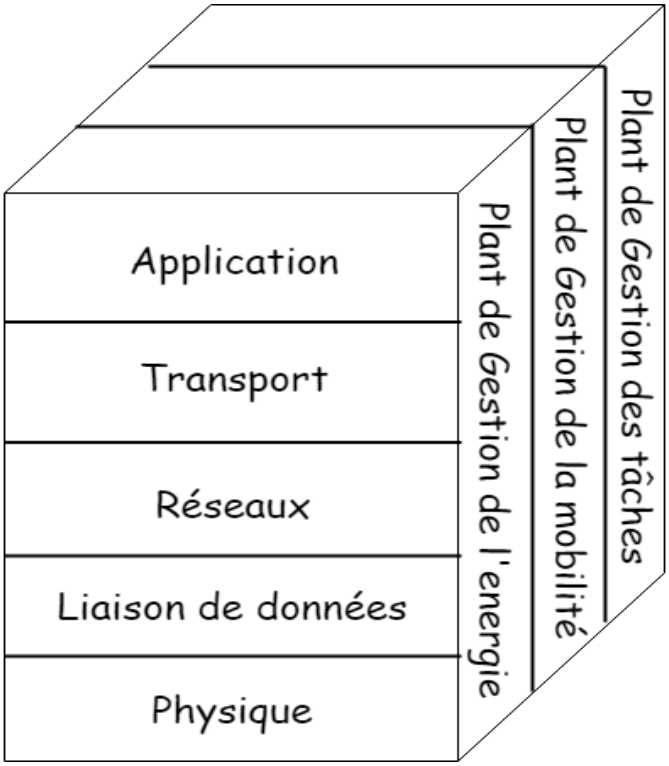

Fig. 1.5 – L'architecture d'un WSN

1.4.1 Les couches WSN OSI

- **La couche physique :** La fonction de cette couche est d'assurer la fiabilité et d'éviter la congestion, où beaucoup de protocoles conçus pour assurer cette fonction sont soit appliquée en amont ou en aval. Ces protocoles utilisent des mécanismes différents pour la détection des pertes et la récupération des pertes [7, 8]. Cette couche est particulièrement nécessaire lorsqu'un système est organisé pour accéder à d'autres réseaux.

- **La couche liaison de données :** Cette couche a pour rôle le multiplexage des flux de données, le partage de l'accès au medium et le contrôle d'erreur. Elle assure une connexion point a point ou point a multipoint fiable dans une communication réseau. [10, 11] Elle est composée de deux couches : la couche LLC (Contrôle des Liens Logiques) et la couche MAC (Contrôle d'accès au medium). Les protocoles de la sous couche MAC sont appelés à effectuer d'importantes opérations citant :

- Etablir des liens de communication entre les nœuds capteurs voisins.
- Fournir une fiabilité entre ces nœuds voisins.
- Partager équitablement les canaux de communication entre les nœuds du réseau.

- **La couche réseau :** La fonction principale de cette couche est le routage. Cette couche présente de nombreux défis en fonction de l'application, mais apparemment, les principaux problèmes sont les suivants : économie d'énergie, mémoire et tampons limités, le capteur n'a pas d'ID global et doit être auto-organisé. C'est différent des réseaux informatiques avec adresse IP et dispositif central de commande [12, 13].

- **La couche transport :** Cette couche est responsable du maintien des flux de données dans les applications utilisées et de la sauvegarde des données dans le cache des capteurs. Elle est particulièrement nécessaire pour accéder au réseau de capteurs par le biais d'un réseau externe comme l'internent [14]. A l'instar des autres couches, de nouvelles approches doivent être mises en place pour faire face aux contraintes inhérentes à ce type de réseau.

- **La couche application :** Cette couche est responsable de la gestion des trafic. Elle permet d'envoyer des requête pour récupérer certaines informations ainsi de fournir des logiciels pour toutes les applications qui traduisent les données sous une forme compréhensible.

1.4.2 Les plans de gestion de la pile protocolaire :

En plus des cinq couches, la pile protocolaire dans un réseau de capteurs comporte trois plans de gestion de l'énergie ou couches transversaux qui sont : plan de gestion de l'énergie, plan de gestion de la mobilité et plan de gestion des tâches. Ces couches servent à gérer le réseau et à faire en sorte que les capteurs fonctionnent ensemble afin d'accroître l'efficacité globale du réseau[12].

- **Le plan de gestion d'énergie :** Ce plan gère la consommation d'énergie par les capteurs.

- **Le plan de gestion de la mobilité :** Ce niveau enregistre le mouvement des nœuds capteurs et détecte leurs position, permet de maintenir l'itinéraire d'un capteur vers un utilisateur et garde la trace de l'emplacement de ses voisins.
- **Le plan de gestion des taches :** S'occupe de la répartition des taches pour une région donnée.

1.5 Domaines d'application des réseaux de capteurs

Les WSNs trouvent leurs intérêts dans des applications diverses et innovatrices. Le champs d'application des réseaux de capteurs est devenu très large grâce à la diminution des coûts de fabrication des micro-capteurs,leurs taille réduite, l'absence des cables entre les nœuds, l'élargissement de la gamme des types de capteurs disponibles (thermique, optique, vibrations, etc.) et l'évolution des support de communication sans fil.

1.5.1 Domaine militaire

Comme dans le cas de plusieurs technologies, le domaine militaire a été un moteur d'origine pour le développement des réseaux de capteurs. Le déploiement rapide, le coût réduit, l'auto- organisation et la tolérance aux pannes des réseaux de capteurs sont des caractéristiques qui rendent ce type de réseaux un outil appréciable dans un tel domaine. [7] Un réseau de capteurs déployé sur un secteur stratégique ou complexe d'accès, permet par exemple la surveillance du champ de bataille, détection des attaques Nucléaires, Biologiques et chimiques, estimation des dommages ainsi que la reconnaissance des forces opposées et du terrain.

1.5.2 Domaine médicale

Les réseaux de capteurs sont largement répandus dans le domaine médical, incluant des applications comme : collecter des informations de meilleure qualité facilitant ainsi le diagnostic de quelques maladies et aussi l'intervention rapide si les mesures effectuées par les capteurs sont anormales,

surveiller des fonctions vitales d'un organisme vivant pourrait à l'avenir être facilitée par des micro-capteurs avalés ou implantés sous la peau ,fournir une interface d'aide pour les handicapés, collecter des informations physiologiques humaines de meilleure qualité, surveiller en permanence les malades et les médecins à l'intérieur de l'hôpital.

1.5.3 Domaine environnementale

Dans ce domaine, les capteurs peuvent êtres exploités pour suivre du déplacement des oiseaux, de petits animaux et d'insecte,détecter les catastrophes naturelles (feux de forêts, tremblements de terre, etc.), connaître la qualité de l'air et détecter la pollution, détecter des fuites des produits toxiques (gaz, produits chimiques, pétrole, etc.) dans des sites industriels tels que les centrales nucléaires et les pétrolières. Les réseaux capteurs peuvent même être utilisé dans la recherche météorologique et géophysique.

1.5.4 Domaine commercial

Les réseaux de capteurs ont prouvé leurs utilité dans le domaine commercial. Dans ce domaine on peut énumérer plusieurs applications citant : contrôler le stockage des produit ainsi que leurs livraison (concernant la chaîne du froid), suivre le procédé de production d'un produit (tous les étapes), contrôler et automatiser le processus d'usinage, ect. Pour amélioré la qualité de service d'une entreprise en terme de livraison on peut même offrir à un client la possibilité de suivre le paquet qu'il attend en temps réel et connaître la position de ce dernier.

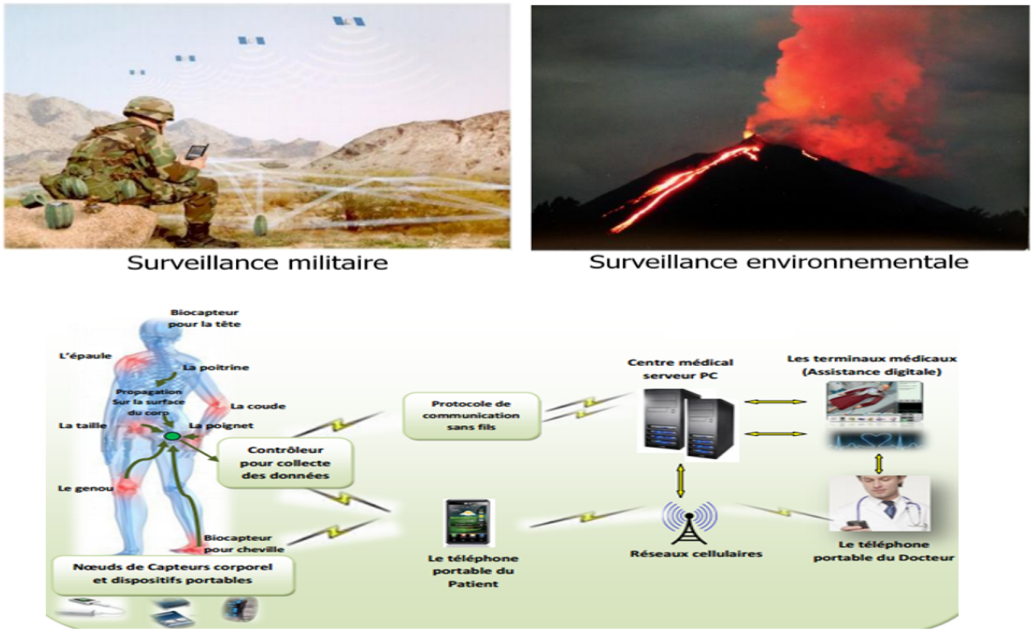

Fig. 1.6 – Les cas d'utilisation d'un réseau de capteurs sans-fil

1.6 Le problème majeur d'un réseau de capteurs sans fil et le challenge :

L'un des défis auxquels est confronté un réseau de capteurs sans fil (WSN) est la dissipation d'énergie des nœuds ayant une incidence sur la durée de vie de la batterie du nœud. Cela est dû au fait que les nœuds plus proches de nœud puits supportent des charges de trafic plus lourdes que tout autre nœud. Par conséquent, les nœuds autour du nœud puits épuiseraient leur énergie plus rapidement ce qui cause par la suite ce qu'on appel un trou d'énergie. Après cela, les données ne peuvent plus être transmises, même s'il reste encore de l'énergie dans les nœuds de la région externe, ce qui affecte la durée de vie du réseau. Comme les réseaux de capteur sont utilisés pour des tâches critique, pensé à l'optimisation ou à la conservation de l'énergie pour chaque nœud capteur est une conséquence très probable. Dans la littérature, le concept d'ensemble dominant connecté [15–18] a été étudié pour la construction d'une ossature de routage avec une consommation d'énergie minimale dans les WSNs. Ces papiers considèrent le poids sur chaque nœud au lieu du poids sur chaque bord. Ceci a conduit à l'introduction du problème de l'arbre dominant DTP [5, 6] avec l'objectif de

prolonger la durée de vie d'un capteur. [19] Le DTP semble résoudre le problème de la consommation de l'énergie mais ce dernier et gourmand en terme de temps d'exécution. Pour ce problème, les chercheurs ont pensé à l'approximation de ce problème en un problème polynomial et ils ont développé des algorithmes d'optimisation qui visent à construire un arbre dominant dans un temps minimal, nommé les méthodes approchées, tous ces détails seront exploité dans les prochains chapitres.

1.7 Conclusion :

Un réseau de capteurs est constitué de plusieurs nœuds capteurs capables de collecter et de traiter des informations sur l'environnement dans lequel ils sont déployés et de les transmettre vers un site donné (station). Ils se caractérisent par une grande flexibilité et tolérance aux fautes. Grâce à ces caractéristiques le champ d'application des réseaux de capteurs s'est étendu pour contenir la plupart des domaines tels que l'industrie, la recherche, l'environnement, la médecine ainsi que le domaine militaire. Cependant, L'apparition de ce type de réseaux a engendré plusieurs défis notamment pour la résolution du problème de trou d'énergie. Plusieurs solution ont été proposer pour cela, mentionnant le calcule de l'arbre dominant, un problème Np-Hard, nécessite pour sa résolution les méta-heuristiques qui sont considérées comme les meilleurs approches d'optimisation. des détails explicatifs font l'objet de notre prochain chapitre.

2eme chapter

2.1 Introduction

L'optimisation combinatoire occupe une place très importante dans divers domaines. En effet, elle définit un cadre formel pour de nombreux problèmes dans plusieurs secteurs tels que l'industrie, la finance ou tout simplement les problèmes de la vie quotidienne. La solution optimale à un problème d'optimisation ne peut que très rarement être déterminée en un temps polynomial. Il est donc souvent nécessaire de trouver des modes de résolution qui fournissent une solution de bonne qualité dans un laps de temps raisonnable. Il existe donc des méthodes de résolution exactes qui sont caractérisées par le fait qu'elles permettent d'obtenir une ou plusieurs solutions optimales, ainsi que des méthodes de résolution approchées qui fournissent des solutions de bonne qualité (proches de l'optimal mais sans garantie d'optimalité) et dont le temps de résolution sera plus faible [20].

Ce chapitre sera structuré comme suit : d'abord, nous allons présenter brièvement les techniques de routage dans les RCSFs, par la suite nous nous pencherons sur les techniques d'optimisation combinatoire, exactes et approchées surtout, pour la résolution des problèmes NP-Difficiles. Tout ceci sera clôturé par une conclusion.

Depuis une vingtaine d'années, les heuristiques les plus populaires, et également les plus efficaces, sont des techniques générales, appelées méta-heuristiques, qu'il s'agit de l'adapter à chaque problème dont sa complexité est NP-hard. Dans ce chapitre, nous allons définir c'est quoi une méta-heuristique, la classification des méthodes utilisées dans les méta-heuristiques.

2.2 Techniques d'optimisation combinatoire :

L'importance de l'optimisation combinatoire se justifie d'un coté par la grande difficulté des problèmes d'optimisation et d'un autre coté par la quantité innombrable d'applications pratiques pouvant être formulées sous la forme de tels problèmes. En effet, ces derniers sont souvent faciles à formaliser ou à exprimer, mais peuvent toutefois, être très difficiles à

résoudre. Néanmoins, la plupart d'entre eux ne possèdent pas à ce jour de solution efficace valable pour toutes les données. [21]

Un problème d'optimisation combinatoire est exprimé sous forme d'une fonction objectif qui est à maximiser, ou à minimiser, selon le type de problème traité. La résolution d'un tel problème revient à chercher la meilleure solution (optimum global) dans l'ensemble des solutions réalisables, sans pour autant être coincé dans des solutions intermédiaires (optimum locaux), propres à un sous-espace de recherche (voir Figure 2.1).

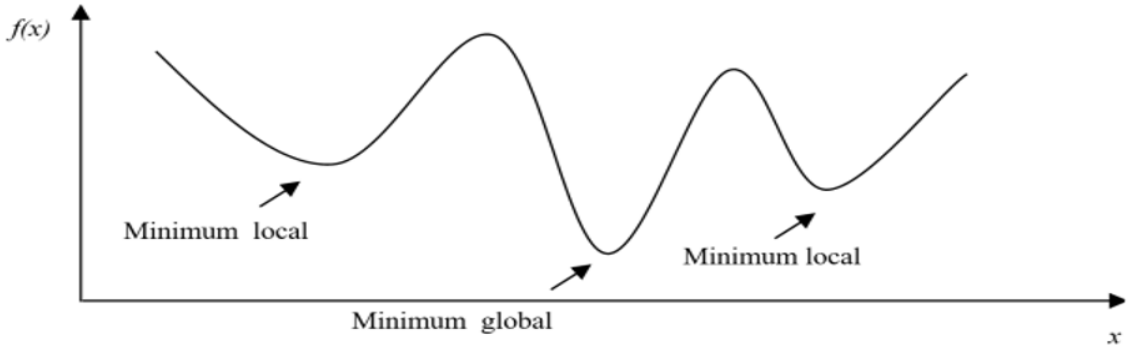

Fig. 2.1 – Différence entre un optimum global et des optima locaux

L'élégance des problèmes d'optimisation combinatoire réside dans la possibilité de les modéliser en problèmes de la théorie des graphes et ainsi profiter des outils de cette dernière.

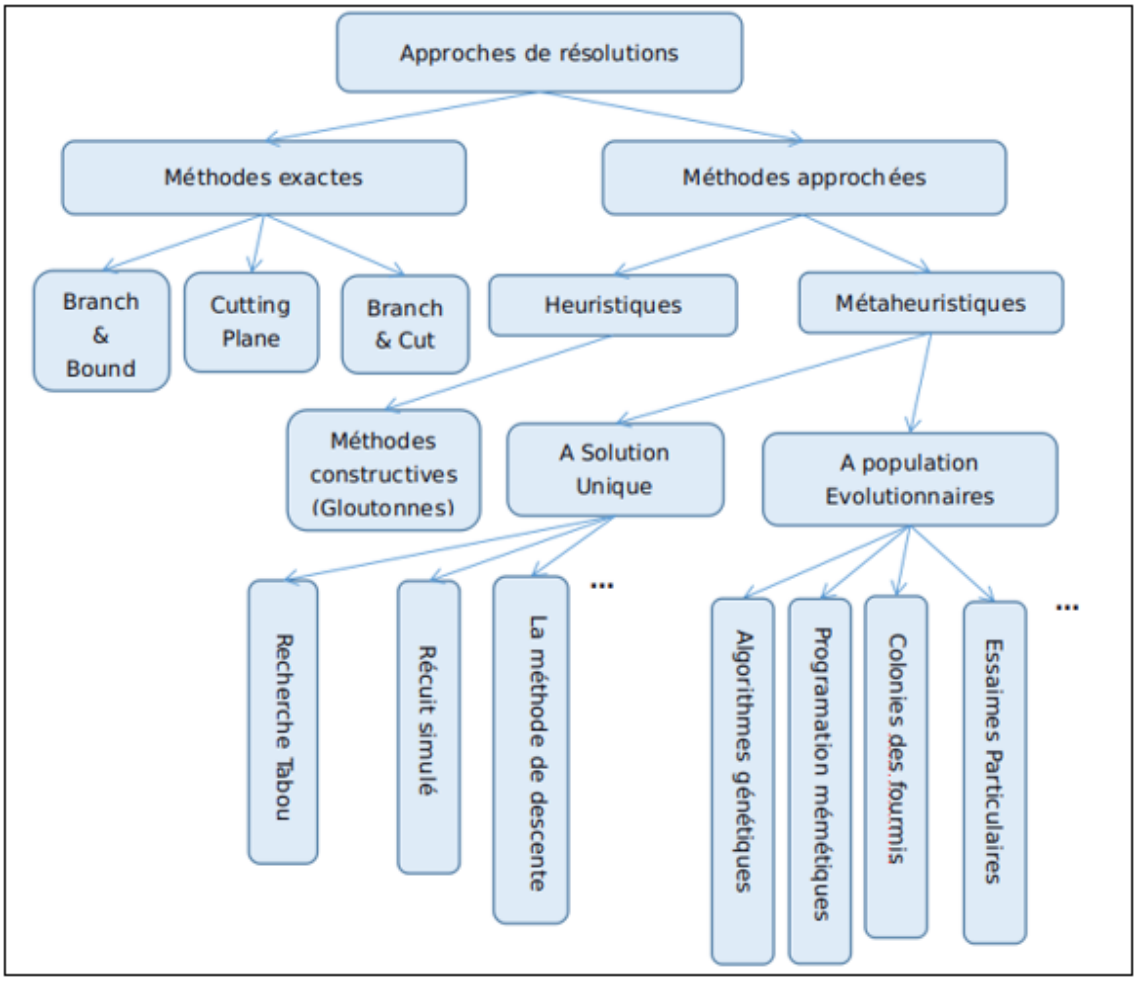

Fig. 2.2 – Classification des méthodes de résolution d'optimisation combinatoire

De nombreuses méthodes de résolution ont été développées par la communauté scientifique (recherche opérationnelle, intelligence artificielle).

Ces méthodes sont divisées en deux grandes catégories : les méthodes exactes basées sur l'énumération de toutes les solutions réalisables, chose qui garantie la complétude de la résolution, et les méthodes approchées qui perdent en exactitude pour gagner en efficacité [16] . La figure 2.2 illustre un schéma résumant la classification des différentes méthodes discutées le long du chapitre.

2.2.1 Méthodes exacte

Ces méthodes sont dites complètes ou exactes car elles permettent de trouver la solution optimale pour une instance de taille finie dans un temps

limité et de prouver son optimalité [22] . Ces méthodes se basent généralement sur sur une recherche complètes de l'espace des combinaisons afin de trouver une solution optimale.
Les algorithmes exacts les plus réussis dans la littérature :

a) **La méthode séparation et évaluation (Branch and Bound) :** elle repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations. Le branch-and-bound est basé sur trois axes principaux : L'évaluation, la séparation, la stratégie de parcours.

   - **L'évaluation :** permet de réduire l'espace de recherche en éliminant quelques sous ensembles qui ne contiennent pas la solution optimale.
   - **La séparation :** a pour but de choisir un sous-problème parmi tous ceux non encore choisis. Elle associe donc à chacun une évaluation (valeur minimale) de toutes les solutions le constituant. Un sous-problème peut être supprimé dans le cas où son évaluation est supérieure à la meilleure solution connue.
   - **La stratégie de parcours :** c'est le processus qui permet de parcourir l'ensemble des sommets et de déterminer lequel sera séparé.

b) **Les méthodes de coupes planes (Cutting-Plane) :** type de problème à résoudre. Néanmoins, ce sont des méthodes destinées à trouver des solutions entières pour des problèmes d'optimisation combinatoire qui sont représentés sous forme d'un programme linéaire [23] .

c) **La méthode (Branch and Cut) :** face aux problèmes difficiles. De même que l'algorithme du "Branch and Bound". Pour cela la méthode "Branch and Cut" qui conjugue l'effort des deux algorithmes précédemment cités est utilisée [24, 25] .

2.2.2 Les méthodes approchées :

Ces méthodes s'appliquent sur tous les problèmes peut importe leurs complexités et vise à trouver une solution admissible en un temps raisonnable, mais ne garantissent pas l'optimalité d'un solution. En outre, elles

ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires. Elles englobent deux classes : Heuristiques et Méta- heuristiques :

2.2.2.1 Heuristiques :

Les heuristiques sont des règles empiriques simples basées sur l'expérience, ne fournissant pas nécessairement une solution optimale. L'avantage d'utiliser une heuristique réside dans l'efficacité de calculer une solution approchée dans un temps raisonnable et ainsi accélérer le processus de résolution exact, qui peut s'avérer long pour des problèmes à large échelle. Généralement, une heuristique n'offre aucune garantie quant à la qualité de la solution. On peut distinguer une classe d'heuristique qui est les méthodes constructives.

Les approches constructives construisent une ou plusieurs combinaisons de façon incrémentale, c'est-à-dire, en partant d'une solution initiale vide, et à chaque itération, une variable est choisie (selon une heuristique ou aléatoirement) jusqu'à l'obtention d'une combinaison complète. c'est pour cela ces approches sont dites "basées sur les modèles" dans [26] .
Il existe différentes stratégies pour choisir les composants à ajouter à chaque itération, les plus connues étant les stratégies gloutonnes. Ces stratégies consistent à construire une solution pas à pas sans retour arrière, en prenant à chaque étape la solution qui semble la meilleure localement (selon une heuristique), en espérant obtenir une solution optimale.

2.2.2.2 Meta-Heuristiques :

Une Méta-heuristique peut être définie comme une méthode algorithmique capable de guider et d'orienter le processus de recherche dans un espace de solution (souvent très grand) à des régions riches en solutions optimales dans le but de trouver des solutions, peut-être pas toujours optimales, en tout cas très proches de l'optimum, en un temps raisonnable.

2.2.2.3 Méthodes de voisinage (A solution unique) :

Les méthodes de voisinage se basent sur la notion de voisinage. Elle a plusieurs méthodes chacune débute avec une configuration initiale, et réa-

lise ensuite un processus itératif qui consiste à remplacer la configuration courante par l'un de ses voisins en tenant compte de la fonction de coût. Ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt concerne généralement une limite pour le nombre d'itérations, le temps d'exécution ou un objectif à réaliser. Présentant quelques méthodes :

a) **Recuit simulé :** La recherche recuit simulé a été introduite en 1983 par Kirkpatrick et al.[24]. Cette méthode originale est basée sur les travaux bien antérieurs de Metropolis et al. [27] et elle est considérée comme la plus ancienne méta-heuristique.
Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique.
La popularité du recuit simulé a été incontestable pendant des années. D'abord cette méthode est facile à implémenter et elle a permis de résoudre de nombreux problèmes NP-difficiles [28, 29].

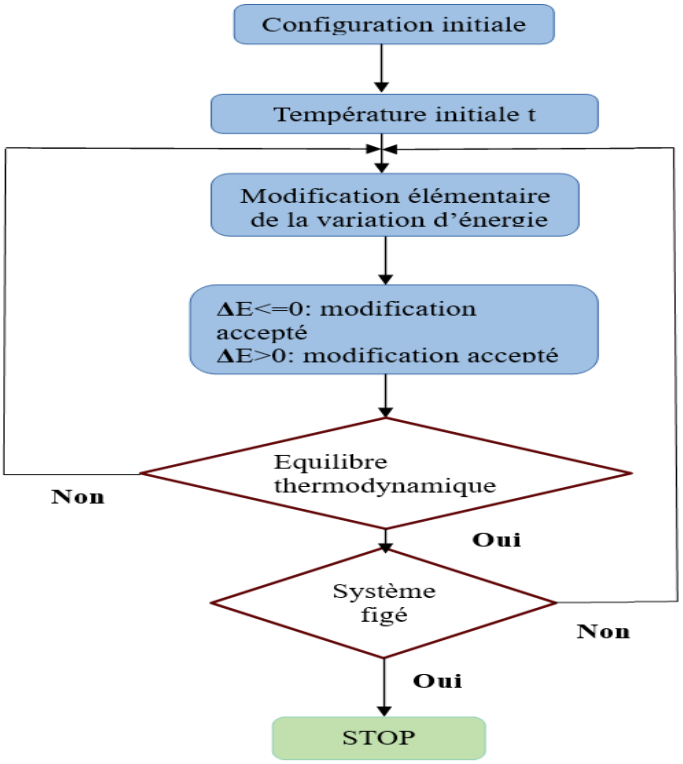

Fig. 2.3 – Organigramme illustrant le fonctionnement de la recherche Recuit simulé

Algorithme 1 : Recuit simulé

$t \leftarrow 0$, Initialiser la température T en fonction du schéma de refroidissement
$s \leftarrow s_0$, une solution initiale
$Best \leftarrow S$
tant que *(Best est la meilleure)* $\cap$ *(nombre iter=max)* $\cap$ $(t \leq 0)$ **faire**
  $r \leftarrow s'$ où $s' \in N(s)$
  $\rho \leftarrow random(0.1)$
  $\triangle E = f(r) - f(s)$
  si $f(r) < f(s) \cup \rho < e^{-\triangle E/t}$ **alors**
    $s \leftarrow r$
  fin
  si *f(s)<f(Best)* **alors**
    $Best \leftarrow s$
  fin
  Décrémenter le t
fin

b) **Recherche tabou :** La méthode de recherche avec tabous, ou simplement recherche tabou (TS :Tabu Search) a été formalisée par Fred Glover en 1986 [30] . Elle utilise explicitement l'historique de la recherche, à la fois pour échapper aux minimaux locaux et pour mettre en œuvre une stratégie d'exploration. Sa principale caractéristique est en effet basée sur l'utilisation de mécanismes inspirés de la mémoire humaine. A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine s' $\in N(s)$ à chaque itération, la recherche tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si s' est plus mauvaise que s. La recherche tabou ne s'arrête donc pas au premier optimum trouvé.
La méthode TS utilise une liste tabou, qui mémorise les dernières solutions rencontrées (ou des caractéristiques de solutions) vers lesquelles il est interdit de se déplacer. Ce procédé simple de mémoire permet de choisir le meilleur voisin non tabou, même si celui-ci dégrade la fonction-objectif f. Cependant, dans certains cas, les interdictions occasionnées par la liste tabou peuvent être jugées trop radicales. En effet, on risque d'éliminer (en les rendant tabous), certains mouvements particulièrement utiles. Pour éviter cela, on incorpore dans l'algorithme un mécanisme d'aspiration qui détermine des critères selon lesquels un mouvement, bien que tabou, peut quand même être accepté, s'il permet d'obtenir une meilleure solution que toutes celles déjà parcourues.
La taille de la liste tabou contrôle la mémoire du processus de recherche. Pour favoriser l'intensification, il suffit de diminuer la taille de la liste tabou. En revanche, augmenter la taille de la liste tabou, forcera le processus de recherche à explorer des régions plus vastes, favorisant ainsi la diversification. La taille de la liste tabou peut être modifiée au cours de la recherche [31].
Une autre amélioration intéressante de la TS est l'utilisation de structure de mémoire à moyen et à long terme afin d'approfondir les notions d'intensification et de diversification.

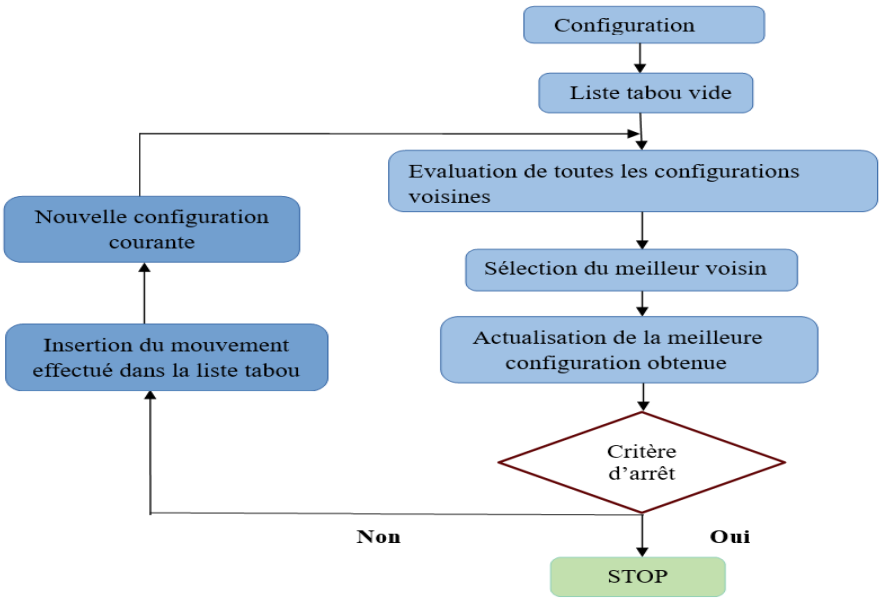

Fig. 2.4 – Organigramme illustrant le Fonctionnement de l'algorithme TS

Algorithme 2 : Recherche Taboue

$l \leftarrow longueurmaximaledelalistetaboue$
$n \leftarrow numberoftweaksdesiredtosamplethegradient$
$S \leftarrow solutioninitial$
$Best \leftarrow S$
$L \leftarrow \{\}$
Insérer s dans L
tant que *(Best est la meilleure)* $\cap$ *(nombre iter=max)* **faire**
  si $longueur(L) > l$ **alors**
    Défiler un élément de la liste L
  fin
  $r \leftarrow r'$ où $r' \in N(s)$
  pour $i \leftarrow 1$ **à** $N - 1$ **faire**
    $w \leftarrow w'$ où $w' \in N(s)$
    si $w \notin L \cap (f(w) < f(r) \cup r \in L)$ **alors**
      $e \leftarrow w$
      si $r \notin L \cap f(r) < f(s)$ **alors**
        $s \leftarrow r$
        Enfiler r dans L
      fin
      si $f(s) < f(Best)$ **alors**
        $Best \leftarrow s$
      fin
    fin
  fin
fin

c) **Recherche locale :** Pour cette méthode de recherche il suffit de tester itérativement de nouvelles solutions potentielles dans la région de la solution courante, et de prendre la meilleure dans le voisinage. La méthode est une généralisation de la méthode de la descente de gradient elle consiste à partir d'une solution s à choisir une solution s' dans un voisinage de s, noté N(s). La nouvelle solution choisie est meilleure que la précédente sous la fonction objective. Cela nous permet d'explorer l'espace des combinaisons de proche en proche, en partant d'une combinaison initiale et en sélectionnant à chaque itération une combinaison voisine de la combinaison courante, obtenue en lui appliquant une transformation élémentaire jusqu'à la convergence à un optimum local. L'algorithme décrivant le principe général de la recherche locale :

Algorithme 3 : Recherche Local

$s \leftarrow$ *solutioninitial*
tant que *(Best est la meilleure)* $\cap$ *(nombre iter=max)* **faire**
  $r \leftarrow s'$ où $s' \in N(s)$
  si $f(r) < f(s)$ **alors**
    $s \leftarrow r$
  fin
fin

d) **Recherche à voisinages variables(VNS) :** La recherche à voisinage variable (VNS : Variable neighborhood search) est une métaheuristique proposée par Hansen et Mladenovic en 1997 [28,29]. Elle est basée sur le principe de changement systématique de voisinage durant la recherche (la performance des méthodes de descente). La procédure de VNS se compose de trois étapes : perturbation (shaking), recherche locale et déplacement. Cet algorithme est efficace si les structures de voisinage sont complémentaires en ce sens qu'un minimum local pour un voisinage n'en n'est pas nécessairement un pour un autre, ce qui veut dire simplement d'utiliser plusieurs voisinages successifs quand on se trouve bloqué dans un minimum local.

Algorithme 4 : Recherche à voisinages variables (VNS)

$s \leftarrow solutioninitial$
répéter
  générer s' ,tel que $s' \in N(s)$
  $s'' \leftarrow Appel$(Recherche Locale)
  si $f(s'') < f(s)$ **alors**
    $s \leftarrow s''$
    $K \leftarrow 1$
  sinon
    $K = K + 1$
  fin
jusqu'à $K = K_{Max}$

2.2.2.4 Méthodes évolutives (A population) :

Depuis le début des années 90, une autre famille d' heuristiques est devenue très populaire : les Méthodes Évolutives [32] , s'inspirant de la théorie de l'évolution «darwinienne» tels que le croisement, la mutation et la sélection pour résoudre des problèmes divers. Contrairement à la Recherche Locale qui tente d' améliorer itérativement une solution courante. Les Méthodes Évolutives travaillent sur une population de solutions en appliquant un processus cyclique composé d' une phase de coopération et d' une phase d'adaptation individuelle qui se succèdent à tour de rôle. Dans la phase de coopération, les solutions de la population courante sont comparées entre elles, puis combinées, dans le but de produire de nouvelles solutions qui héritent des bons aspects de chaque membre de la population. Dans la phase d'adaptation individuelle, chaque solution dans la population peut évoluer de manière indépendante. On peut utiliser le même type de critère d' arrêt que dans la Recherche Locale, ou alors on peut décider de stopper une Méthode Évolutive dès que les solutions dans la population sont jugées trop similaires. Une description générale des Méthodes Évolutives est donnée ci-dessous :

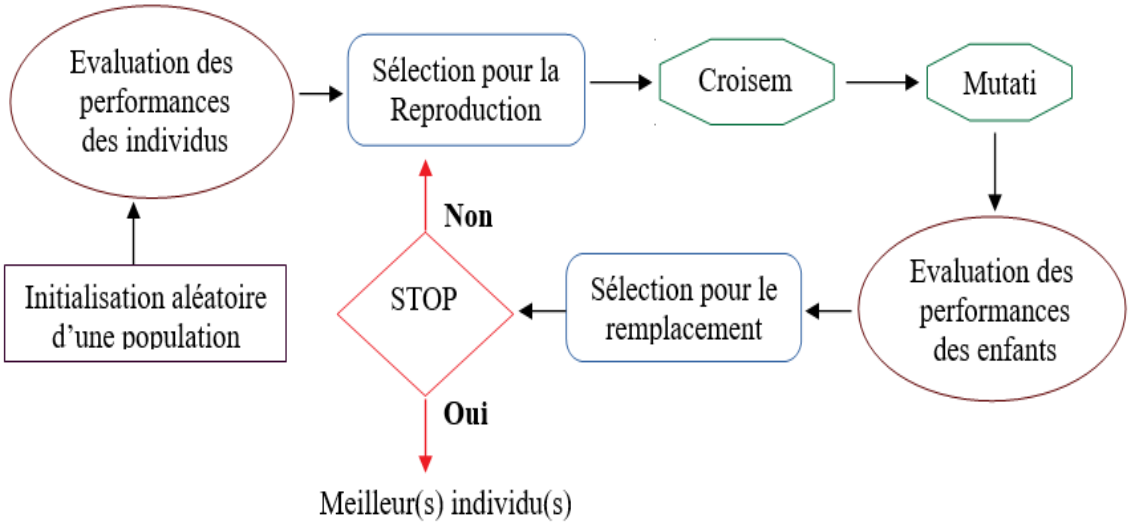

Fig. 2.5 – Principe d'un algorithme évolutionnaire (EA) [3].

Le terme Evolutionary Computation englobe une classe assez large de méta-heuristiques qui ont été développés au début des années soixante, telles que les algorithmes génétiques [33], les stratégies d'évolution [34] , la programmation évolutive [35], et la programmation génétique [36]

a) **Algorithmes génétiques(AG) :** développées par J. Holland en 1975 comme outils de modélisation de l'adaptation et qui travaillent dans un espace de chaînes de bits. Elles ont été largement utilisé et développé par D.E. Goldberg en 1989. Cette classe s'inspire de la théorie de l'évolution et des règles de la génétique qui expliquent la capacité des espèces de s'adapter à leurs environnement à l' aide de l' opérateur de mutation, alors que l' échange d'information est gouverné par un opérateur de reproduction et un opérateur de combinaison (ou croisement). L'algorithme ci-dessous décrit ce principe général, dont les principales étapes sont détaillées ci-après.

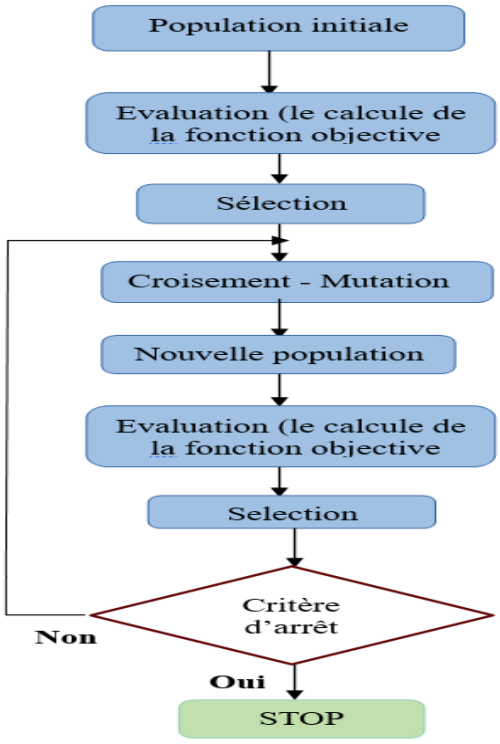

Fig. 2.6 – Organigramme d'un algorithme génétique

Algorithme 5 : Algorithme génétique

Résultat : la meilleure combinaison ayant appartenu à la population

Initialiser la population avec un ensemble de combinaisons de E

tant que *critères d'arrêt non atteints* **faire**

  Sélectionner des combinaisons de la population

  Créer de nouvelles combinaisons par croisement et mutation

  Mettre à jour la population

fin

- **Initialisation de la population :** en général, la population initiale est générée de façon aléatoire, selon une distribution uniforme assurant une bonne diversité des combinaisons. Une solution initiale est de bonne qualité, celle qui nous permet de converger vers un optimum global dans le plus court délai possible.

  Il est à noter que cette méthode est utilisée dans toutes les méta-heuristiques avec population.

- **Sélection :** cette étape consiste à choisir les combinaisons de la population qui seront ensuite croisées et mutées. Il s'agit là de

favoriser la sélection des meilleures combinaisons, tout en laissant une petite chance aux moins bonnes combinaisons. Il existe de nombreuses façons de procéder à cette étape de sélection. Par exemple, la sélection par tournoi consiste à choisir aléatoirement deux combinaisons et à sélectionner la meilleure des deux (ou bien à sélectionner une des deux selon une probabilité dépendant de la fonction objectif).

- **Croisement :** est une opération de diversification. cette opération consiste à générer de nouvelles combinaisons, à partir des deux individus sélectionnés. Là encore, il existe de nombreux opérateurs de croisement. Une méthode simple consiste à choisir aléatoirement un point de croisement, à couper chaque combinaison parente en ce point, puis à reformer deux enfants en échangeant les parties composant les parents de part et d'autre du point de croisement.
- **Mutation :** cette opération consiste à modifier de façon aléatoire certains composants des combinaisons obtenues par croisement.
- **Mise à jour de la population :** cette étape consiste à remplacer certaines combinaisons de la génération précédente par certaines combinaisons issues des opérations de croisement et de mutation, formant de la sorte une nouvelle génération. Là encore, il existe différentes stratégies de remplacement, favorisant plus ou moins la diversité, et plus ou moins élitistes. On peut par exemple choisir de ne garder que les meilleurs individus, qu'ils soient issus de la nouvelle génération ou de l'ancienne, ou bien ne garder que les individus de la nouvelle génération, indépendamment de leur qualité.
- **Critères d'arrêt :** le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'une combinaison de qualité suffisante soit générée, ou bien jusqu'à ce qu'une limite de temps soit atteinte. On peut également utiliser des indicateurs de diversité (comme par exemple le taux de ré-échantillonnage ou la distance pair-à-pair) pour arrêter le processus lorsque la population est devenue trop uniforme. On peut aussi limiter le nombre d'itération possible ou encore les probabilités d'application des opérateurs de croisement et de mutation.

b) **L'optimisation des colonies de fourmis (ACO) :** est une technique d'intelligence artificielle qui s'inspire du comportement intelligent des fourmis lors de la recherche de nourriture. Cette métaheuristique permet de trouver des solutions de haute qualité aux problèmes d'optimisation combinatoire.

En raison de leur comportement de recherche de nourriture, les fourmis trouvent les chemins les plus courts entre leur nid et leur source de nourriture, en déposant sur la terre une substance chimique appelée phéromone. Cela forme une traînée de phéromone qui est utilisée pour la communication stigmergique. La présence d'une telle phéromone sur les trajectoires affecte la prise de décision des fourmis concernant les trajectoires choisies par elles. Les fourmis sélectionnent de manière probabiliste les chemins marqués par de fortes concentrations de phéromones ce qui permet aux fourmis de trouver le chemin le plus court vers la source de nourriture.

Algorithme 6 : L'optimisation des colonies de fourmis (ACO)

Résultat : la meilleure solution trouvée
Initialisation : initialisation de phéromones et les paramètre avec des valeurs constantes
Construction de la solution initiale
tant que *condition d'arrêt non atteinte* **faire**
    tant que *nombre de fourmis non atteint* **faire**
        Construction d'une solution selon la concentration de phéromone
        Mise à jour en ligne de la table des phéromones
    fin
    si *la meilleure solution trouvée par l'itération courante est meilleure que celles des itérations précédente* **alors**
        Mise à jour de l a solution courante
    fin
    Mise à jour hors ligne de la table des phéromones
fin

- **Solution initiale :** en général, la solution initiale est générée de façon aléatoire.Une solution est composée de plusieurs attributs définis selon la nature du problème.

- **Initialisation de la table de phéromone :** la table de phéromone contient la concentration de phéromone pour tous les attributs possible d'une solution. Au départ tous les attributs ont une même concentration de phéromone.
- **Mise à jour de phéromone :**
  - **En ligne :** les attributs communs entre la solution courante et la meilleure solution vont être favoriser par une valeur constante supplémentaire.
  - **Hors ligne :** les attributs appartenant à la meilleure solution vont être favoriser par la valeur constante.

c) **L'algorithme à base de biogéographie (BBO)** BBO est un algorithme inspiré de la biogéographie, qui étudie la répartition géographique des organismes biologiques.

Tout comme GA et PSO qui sont basés sur la biologie, BBO est un algorithme basé sur une population dans lequel chaque population de solutions candidates est utilisée dans la procédure de recherche d'un optima global. BBO présente certaines caractéristiques communes avec l'algorithme d'optimisation, le GA. En GA, un individu de la population s'appelle un chromosome et a sa propre valeur de condition physique. De même, dans BBO, chaque individu est qualifié d'habitat et a son indice de qualité de l'habitat (HSI) pour évaluer sa qualité en tant que solution. Comme nous avons affaire à un problème de minimisation, un habitat à faible indice de sécurité d'impact représente une bonne solution et un habitat à haut indice de faiblesse est plutôt une solution médiocre. Chaque chromosome de GA est constitué de gènes, tandis que, pour BBO, chaque habitat est caractérisé par des variables d'indice de pertinence (SIV). L'AG compte deux opérateurs principaux : le croisement et la mutation. Pendant ce temps, chez BBO, les principaux opérateurs sont la migration et la mutation. L'opérateur de migration est composé d'émigration et d'immigration. Il est utilisé pour améliorer et faire évoluer les habitats (solutions au problème d'optimisation) de la population. Les fonctionnalités de solution (SIV) émigrent d'habitats à faible HSI (habitats d'émigration) vers des habitats à haut HSI (habitats d'immigration). Il existe différentes alternatives pour les processus de migration et de mutation de BBO. La manière dont nous implémentons ces deux opérateurs est

expliquée en détail dans le prochain chapitre.

Algorithme 7 : L'algorithme à base de biogéographie (BBO)

Résultat : la meilleure solution ayant appartenu à la population
Générer aléatoirement un ensemble de solutions initiales (îles)
tant que *le critère d'arrêt n'est pas atteint* **faire**
    Évaluer la fitness (HSI) de chaque solution
    Calculer le nombre d'espèce S, le taux d'immigration λ et d'émigration μ pour chaque solution
    Migration
    Mutation : muter les individus au taux de mutation
    Remplacement de la population par les descendants
    Implémenter l'**élitisme**
fin

Dans le processus de migration (algorithme 8), quand une solution est sélectionnée pour être changée, nous utilisons le taux d'immigration λ pour décider si un SIV sera changé. Dans ce cas, nous utilisons le taux d'émigration μ pour décider quelle bonne solution migrera son SIV.
Cette algorithme sera suivit d'une figure qui ellustre le fonctionnement de ce processus (figure 2.7)

Algorithme 8 : migration

```
pour i ← 1 à N faire
    Utiliser λ_i pour décider, de manière probabiliste, d'immigrer à X_i
    si rand(0,1) < λ_i alors
        pour j ← 1 à N faire
            Sélectionner un habitat d'émigration X_j avec une probabilité αμ_j
            si rand(0,1) < μ_j alors
                Remplacer une variable de décision (SIV) choisie aléatoirement dans par la variable correspondante dans X_j
            fin
        fin
    fin
fin
```

Où :
λ_k : est le taux d'immigration dans un habitat de k espèces
μ_k : est le taux d'émigration dans un habitat de k espèces.
E : est le taux maximal d'émigration.
I : est le taux maximal d'immigration. Supposons que nous avons une population de solutions candidates à un problème, représentées par des vecteurs (habitats $H_i, i = 1...n$)
N : est le nombre maximum d'espèces.
k : est le nombre d'espèces.
Mutation : c'est le même principe qui est utilié pour l'algorithme génétique.
Elitisme : cette méthode consiste à garder les meilleurs individus de chaque population. La méthode d'élitisme risque d'élaguer les individus de mauvaise qualité mais ça n'empêche pas de produire une population avec des résultats meilleurs ou au moins de créer des individus qui auront pu apporter de quoi créer de très bonnes solution dans les populations suivantes.

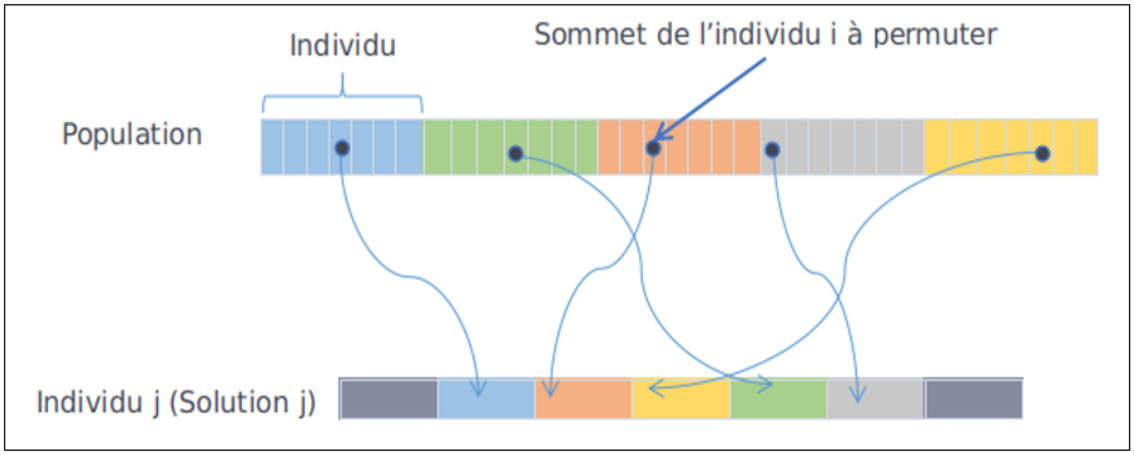

Fig. 2.7 – processus de migration de BBO

d) **Algorithmes mémétiques (algorithme de colonies de fourmis & d'essaim de particules) :** Le terme «algorithmes mémétiques» est apparu pour la première fois dans [38], introduit par Moscato en 1986. L'idée principale de ce genre d'algorithme est de rendre plus agressif un algorithme génétique par l'ajout d'une recherche locale en plus de la mutation.

La faible vitesse de convergence est la première observation provenant de l'implémentation de l'algorithme génétique. Pour cet effet Moscoto a pensé d'ajouter une recherche locale qui peut être une méthode de descente ou une recherche locale plus évoluée (recuit simulé ou recherche tabou par exemple). Cette recherche locale sera appliquée à tout nouvel individu obtenu au cours de la recherche.

D'une vue globale, l'application de cette simple modification n'influe en rien sur le comportement de l'algorithme, mais cela peut apporter de profondes changement en ce dernier. Après avoir créé un nouvel individu à partir de deux parents sélectionnés, on applique une recherche locale et sous certaines conditions on applique un opérateur de mutation à cet individu. Les conditions peuvent être une certaine probabilité. Il est aussi possible d'ajouter un critère d'aspiration ou d'autres techniques plus évoluées à cet endroit.

Algorithme 9 : Simple Algorithme mémétique

Résultat : la meilleure solution ayant appartenu à la population

Initialisation : générer une population initiale P de solutions de taille $|P| = n$

répéter

Sélection : choisissez 2 solutions s et s' avec la technique volue

Croisement : combinez les solutions parentes s et s' pour former une solution enfant y

Recherche locale : appliquer une procédure de recherche locale sur y sous certaines conditions

Mutation : appliquer un opérateur de mutation sur y dans des conditions

Choisir un individu y' à remplacer dans la population

remplacer y' par y dans la population

jusqu'à *critère d'arrêt satisfait*

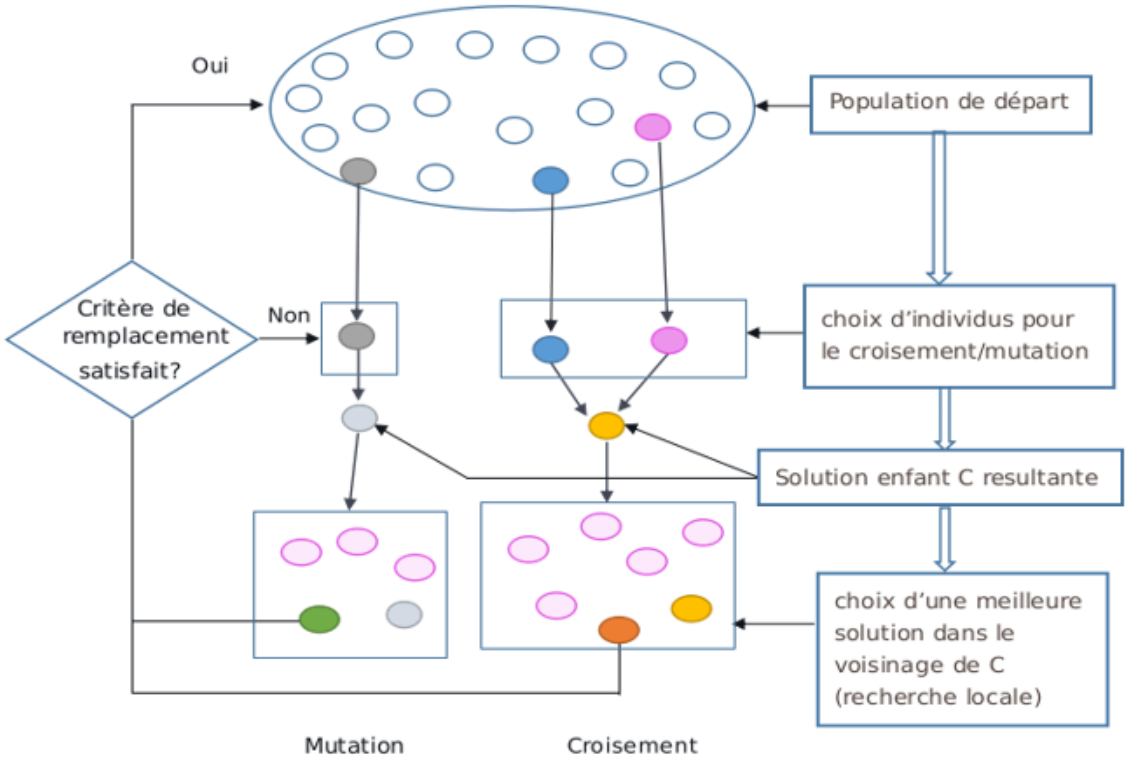

Fig. 2.8 – Schéma explicatif d'un algorithme mémétique

2.3 Conclusion

Nous avons présenté dans ce chapitre l'état de l'art des méta-heuristiques pour l'optimisation des problèmes Np-difficile. Plus de détails sur le problème de l'arbre dominant qui en est un, ainsi que les méthodes utilisées pour sa résolution seront mis au clair dans le prochain chapitre.

3eme chapter

3.1 Introduction

3.2 Définition du problème

3.3 Complexité et approximation

Algorithme 10 : Recuit simulé

Données : un graphe général pondéré non orienté $G = (V, E, w)$
Résultat : Un DT de G
Initialiser une liste pour sauvegarder le DT et son poids
Trouver un sommet factice u en G avec un degré minimum
Transformer G en $G' = (V', E', w')$ en utilisant la technique de transformation
pour *chaque nœud* v *où* $(u, v) \in E$ **faire**
    Exécution de l'algorithme DST [37] dans G et définir v comme racine r pour obtenir un DT
    Sauvegarder le DT et son poids dans la liste
fin
Retourner le DT avec poids minimum dans la liste

Algorithme 11 : Prim

Données : un graphe général pondéré non orienté $G = (V, E, w)$
Résultat : Un arbre couvrant de poids minimal T
`// Initialisation de T`
sommets $\leftarrow$ un sommet de G qu'on choisit
arêtes $\leftarrow$ aucune
répéter
  Trouver toutes les arêtes de G qui relient un sommet de T et un sommet extérieur à T
  Sélectionner l'arête ayant le plus petit poids parmis l'ensemble des arêtes trouvées
  Ajouter à T cette arête et le sommet correspondant
jusqu'à *tous les sommets de G soient dans T*
Retourner T

Algorithme 12 : pseudo-code H_DT

Données : un graphe général pondéré non orienté $G = (V, E, w)$
Résultat : sortie : un arbre dominant $DT \subseteq E$
`// Initialise DT`
$DT \leftarrow \varnothing$
pour *chaque sommet* $i \in V$ **faire**
  Mark[i] $\leftarrow 0$
fin
Calcule les chemins les plus courts entre toutes les paires de sommets de G
Triez toutes les arêtes de E en ordre non décroissant en fonction de leur poids
tant que *tous les sommets de V ne sont pas marqués* **faire**
  Sélectionnez une arête non sélectionnée e_{ij} avec un coût minimal, dont au moins un point d'extrémité n'est pas marqué
  pour *chaque sommet v adjacent à i ou j* **faire**
    Mark[v] $\leftarrow 1$
  fin
  si $DT = \varnothing$ *ou* e_{ij} *est adjacent à une arête dans* DT **alors**
    $DT \leftarrow DT \cup \{e_{ij}\}$
  sinon
    Trouver le chemin le plus court ST reliant DT et $\{i, j\}$
    $DT \leftarrow DT \cup ST \cup \{e_{ij}\}$
  fin
  pour *chaque sommet v en ST* **faire**
    $Mark[v] \leftarrow 1$
    pour *chaque sommet k adjacent à v* **faire**
      $Mark[k] \leftarrow 1$
    fin
  fin
fin
Appliquer la procédure d'élagage sur DT
Reconnectez les sommets dominants de DT en y construisant un arbre couvrent de poids minimal
Appliquer l'élagage sur le DT
Reconnectez les sommets dominants de DT en y construisant un arbre couvrent de poids minimal
Retourner DT

Algorithme 13 : Pseudo-code de ABC

```
Générer ne solutions aléatoires E_1, E_2, ..., E_ne
MeilleureSol ← meilleure solution parmi E_1, E_2, ..., E_ne
tant que La condition de terminaison est non satisfaite faire
    pour Chaque abeille employée i dans ne faire
        E' ← determination_d_une_solution_voisine(E_i)
        si E' est meilleure que E_i alors
            E_i ← E'
        sinon
            si E_i n'a pas changé après un nombre limité d'itérations alors
                Remplacer E_i avec une solution générée aléatoirement
            fin
        fin
        si E' est meilleure que MeilleureSol alors
            MeilleureSol ← E_i
        fin
    fin
    pour Chaque abeille onlooker i dans n_0 faire
        s_i ← méthode de sélection de tournoi binaire (E_1, E_2, ..., E_ne)
        O_ni ← determination_d_une_solution_voisine(E_si)
        si O_ni est meilleure que MeilleureSol alors
            MeilleureSol ← O_ni
        fin
        si O_ni est meilleure que E_si alors
            E_si ← O_ni
        fin
    fin
fin
```

Algorithme 14 : Pseudo-code de détermination d'une solution voisine

Données : Une solution s
Résultat : Une solution voisine s
Créer une copie s' de s
si $u_{01} < q_1$ **alors**
  Appliquer la procédure PDE sur s
sinon
  Appliquer la procédure PANDV sur s
fin
Appliquer la procédure pruning sur s
Reconnecter les sommets dominants de s' en construisant un arbre de poids minimum sur eux (MST)
Appliquer la procédure pruning sur s
Reconnecter les sommets dominants de s' en construisant un arbre de poids minimum sur eux (MST)

Algorithme 15 : Pseudo-code de l'algorithme SSGA

Générer une population (popsize) de solutions $s1, s2, ..., s_pop_size$ aléatoirement
$MeilleureSol \leftarrow$ La meilleure solution de la population
tant que *La condition de terminaison est non satisfaite* **faire**
  si $u_{01} < p_c$ **alors**
    $p_1 \leftarrow$ méthode de sélection de tournoi binaire $s1, s2, ..., s_pop_size$
    $p_2 \leftarrow$ méthode de sélection de tournoi binaire $s1, s2, ..., s_pop_size$
    Enfant $\leftarrow$ Opérateur de Croisement $(p1, p2)$
  sinon
    $p_1 \leftarrow$ méthode de sélection de tournoi binaire $s1, s2, ..., s_pop_size$
    Enfant $\leftarrow$ Opérateur de Mutation $(p1)$
  fin
  si *Enfant est un DT partiel* **alors**
    Appliquer procédure de réparation sur Enfant
  fin
  Appliquer la procédure pruning sur le DT de Enfant
  Appliquer l'algorithme de Prim pour construire un arbre de poids minimal sur le sousgraphe de G induit par l'ensemble des sommets dominants de DT
  Appliquer la procédure pruning sur la DT de Enfant
  si *Enfant est meilleure que* $MeilleureSol$ **alors**
    $MeilleureSol \leftarrow Enfant$
  fin
  Appliquer la politique de remplacement
fin
Retourner $MeilleureSol$

Algorithme 16 : Pseudo-code de la procédure pruning

$R_n \leftarrow \{v : v \in DN \text{ et } |ON(v) \cap DN| = 1 \text{ et } CN(v) \subseteq (\cup_{u \in DN \setminus \{v\}})\}$
tant que $R_n \neq \varnothing$ **faire**
  $DN \leftarrow DN \setminus \{v\}$
  $R_n \leftarrow \{v : v \in DN \text{ et } |ON(v) \cap DN| = 1 \text{ et } CN(v) \subseteq (\cup_{u \in DN \setminus \{v\}})\}$
fin
Retourner DN

Algorithme 17 : Pseudo-code de la solution initiale de EA/G-MP

Initialement tous les nœuds dans V sont colorés en BLANC
$Wn \leftarrow V$ $DT \leftarrow \varnothing$ $DN \leftarrow \varnothing$ $I_u \leftarrow \varnothing$
$v \leftarrow Random(W_n)$
$DN \leftarrow DN \cup \{v\}$
Mettre v NOIR
$nb \leftarrow ON(v) \cap W_n$
$W_n \leftarrow W_n \setminus (nb \cup \{v\})$
Mettre tous les noeuds $\in$ nb GRIS
$I_u \leftarrow nb$
tant que $W_n \neq \varnothing$ **faire**
  Générer un nombre aléatoire u_{01} tel que $0 \leq u_{01} \leq 1$
  si $u_{01} < \phi$ **alors**
    $(v, u) \leftarrow$ argument $v \in DN, \{u \in I_u | ON(u) \cap W_n| \geq 1\} w(v, u)$
  sinon
    $v \leftarrow Random(DN)$
    $u \leftarrow \{u : Random(I_u)$ et $|ON(u) \cap W_n| \geq 1\}$
  fin
  $DT \leftarrow DT \cup \{e_{v,u}\}$
  $DN \leftarrow DN \cup \{u\}$
  Mettre u NOIR
  $I_u \leftarrow I_u \setminus \{u\}$
  $nb \leftarrow \{u : u \in ON(u) \cap W_n\}$
  Mettre tous les noeuds nb GRIS
  $I_u \leftarrow I_u \cup nb$
  $W_n \leftarrow W_n \setminus nb$
fin
Appliquer la procédure pruning sur les nœuds dans DT
Reconnecter les nœuds dans DT via MST
Retourner DT

Algorithme 18 : Pseudo-code de l'initialisation du vecteur de probabilités de EA/G-MP.

$nv \leftarrow$ nombre de solutions initiales contenant le nœud $v, \forall v \in V$;
$pv \leftarrow \dfrac{nv}{Np}$, $\forall v \in V$

Algorithme 19 : Pseudo-code de la mise à jour du vecteur de probabilités de EA/G-MP.

$nv \leftarrow$ nombre de solutions initiales contenant le nœud $v, \forall v \in V$;
$pv \leftarrow (1 - \lambda)pv + \lambda \ \frac{nv}{L} \ , \forall v \in V$

Algorithme 20 : Pseudo-code de GM

```
Mettre la couleur de tous les nœuds dans V à BLANC
DT ← ∅
pour nœud v ∈ V dans un ordre quelconque faire
    Générer un nombre aléatoire r1 tel que 0 ≤ r1 ≤ 1
    si r1 < β alors
        Générer un nombre aléatoire r2 tel que 0 ≤ r2 ≤ 1
        si r2 < pv Et ((v est BLANC) Ou (v est GRIS avec au moins un voisin
         BLANC)) alors
            Trouver le chemin le plus court SP entre le nœud v et un nœud u dans
             DT
            Ajouter toutes les arêtes de SP dans DT
            Colorer en NOIR tous les nœuds dans le chemin SP
            Colorer en GRIS tous les nœuds voisins BLANCs des nœuds sur le
             chemin SP
        fin
    sinon
        si v est un nœud dominant dans m i et v est GRIS avec au moins un
         voisin BLANC alors
            Trouver le chemin le plus court SP entre le nœud v et un nœud u dans
             DT
            Ajouter toutes les arêtes de SP dans DT
            Colorer en NOIR tous les nœuds dans le chemin SP
            Colorer en GRIS tous les nœuds voisins BLANCs des nœuds sur le
             chemin SP
        fin
    fin
fin
Retourner DT
```

Algorithme 21 : Pseudo-code de l'opérateur de réparation

tant que $U\ cn = \varnothing$ **faire**
  $v \leftarrow -argmax\, u \in U\, cn(wd(u) > 0)$
  si $v \neq \varnothing$ **alors**
    Sélectionner un nœud v avec le plus petit index de U cn ;
  fin
  Trouver le chemin le plus court SP entre le nœud v et un nœud u dans DT
  Ajouter toutes les arêtes de SP dans DT
  Colorer en NOIR tous les nœuds dans le chemin SP
  Colorer en GRIS tous les nœuds voisins BLANCs des nœuds sur le chemin SP
  Enlever tous les nœuds NOIRs et GRIS de U cn
fin
Retourner DT

Algorithme 22 : EA/G-MP pour DTP

A la génération $g \leftarrow 0$, une population initiale pop(g) qui consiste en N p solutions est générée aléatoirement

Initialiser le vecteur de probabilités p pour tous les nœuds en utilisant l'Algorithme 18

tant que *La condition de terminaison est non satisfaite* **faire**

Sélectionner les L meilleures solutions de pop(g) pour former un ensemble parent parent(g), puis mettre à jour le vecteur de probabilités p en utilisant l'Algorithme 19

Appliquer l'opérateur de mutation guidée GM une fois sur chacune des M meilleures solutions. Un opérateur de réparation est appliqué sur chaque solution générée, si nécessaire, puis MST, l'opérateur de pruning, MST et l'opérateur de pruning sont appliqués sur chaque solution générée pour améliorer sa fitness. Ajouter toutes les M nouvelles solutions générées avec N p - M meilleures solutions à pop(g) pour former pop(g+1). Si le critère d'arrêt est satisfait, retourner l'arbre dominant avec le poids minimum trouvé jusqu'à présent

$g \leftarrow g + 1$

si *la meilleure solution de la population ne s'améliore pas après S générations* **alors**

réinitialiser toute pop(g), sauf la meilleure solution, puis initialiser le vecteur de probabilités p pour tous les nœuds en utilisant l'Algorithme 18

fin

fin

Algorithme 23 : x

Données : x
Résultat : x

4eme chapter

4.1 Introduction

Dans les chapitres précédents, nous avons introduit différentes techniques utilisées pour la résolution des problèmes d'optimisation combinatoire, notamment les méthodes adoptées pour la résolution du problème de l'arbre dominant DTP, dont une grande majorité a servi de source d'inspiration pour l'élaboration des approches que nous avons proposé et qui seront détaillées le long de ce chapitre. Nous allons dans un premier présenter les structures de données utilisées pour représenter les données du problème ainsi que la solution.

4.2 Représentation du graphe modélisant le problème

Un WSN est un graphe pondéré non orienté $G = (V, E, w)$, où V est l'ensemble des nœuds du réseau, E l'ensemble des arêtes et w un poids non négatif attribué pour chaque arête $e = (u, v)$.

Nous utilisons pour représenter le WSN deux vecteurs, un vecteur de taille $|V|$ contenant des entiers représentant chaque nœud et l'autre est de taille $|V - 1|$ il contient une structure d'arc (nœud 1, nœud 2 et le poids entre les deux nœuds. Nous allons dans ce qui suit présenter le codage de la solution ainsi que la méthode d'extraction du DT à partir de cette solution.

4.3 présentation d'une solution

Nous proposons de représenter une solution en utilisant le codage binaire qui est considéré dans la littérature comme un bon moyen pour manipuler les différents gène (position(bit) dans une solution) d'une solution pour une population donnée. Si le gène est égale à '1', cela veut dire que le nœud correspondant est actif, s'il vaut '0', il est inactif.

4.4 Méthode d'extraction d'un arbre dominant à partir du vecteur solution

Au départ nous avons deux vecteurs $(Vect, DN)$, où, $Vect$ contient tous les nœuds de notre graphe, DN est un vecteur binaire de taille $|V|$ initialement null et contiendra par la suite l'ensemble des nœuds dominant .

L'ajout des nœuds à DN se fait via un processus itératif. A chaque fois un nœud est choisi d'une façon aléatoire, et est ajouté à DN. Ce processus poursuit son fonctionnement tant que les nœuds appartenant à DN ne couvre pas tous le graphe ou ils ne sont pas connectés. On dit que DN couvre le graphe si et seulement si tous les nœuds de notre graphe sont soit dans DN soit adjacent à un autre nœud qui est dans DN.

Pour pouvoir construire un DT, il faut d'abord vérifier que les nœuds actifs dans DN sont connectés entre eux. Dans le meilleur des cas, les nœuds actifs dans DN sont connectés. Dans le cas contraire nous devons les rendre connecté en utilisant une méthode de connection décrite si après.

4.5 La méthode de connection

Soit un vecteur DNC initialement vide qui va contenir par la suite ensemble de nœuds connecté. En premier lieu, nous retirons un nœud aléatoirement de DN et nous l'insérons dans DNC. Puis, d'une façon itérative, nous retirons tous les nœuds de DN qui ont au moins un voisin dans DNC et nous l'insérons dans DNV. En second lieu nous, devons insérer tous les nœuds actifs dans DN dans DNC en utilisant une autre fonction. Cette fonction consiste au départ, à choisir un nœud v aléatoirement dans V qui n'est pas dans DNC. Si v a au moins deux voisins un dans DN et l'autre dans DNC, alors nous retirons son voisins dans DN et nous l'insérons avec v dans DNC. Si aucun nœud n'est trouvé nous procédons à une sélection par pair de nœuds où chaque nœud entre eux a soit un voisin dans DN ou dans DNC et nous refaisions le même processus de recherche daans le cas d'un seul nœud à selectionner. Nous ré-exécutons ce prceossus jusqu'à DN devient null.

Une fois que l'ensemble DN soit null, nous appliquant la méthode d'éla-

gage et MST décrit dans le chapitre 3. Nous aurons au final, un arbre dominant DT, représenté sous forme d'un vecteur de type arc (chaque élément de ce vecteur est composé d'un noeud1, noeud2, et le poids entre ces deux nœuds).

4.6 Approche proposée

Dans ce mémoire nous avons proposé plusieurs approches pour résoudre le DTP. Nous avons proposé de le résoudre dans un premier temps en utilisant des méta-heuristiques, et en utilisant la coopération de ces méta-heuristiques dans un second temps. Nous allons structurer ce qui suit en deux parties. Dans la première partie, nous présentons chacune des méta-heuristiques utilisées, dans la seconde partie, nous présentons l'approche coopérative.

4.6.1 description des méta-heuristiques utilisées

La méthode de résolution du problème de l'arbre dominant que nous avons proposée consiste à intégrer un processus de recherche locale à la méta-heuristiques. Nous allons dans ce qui suit décrire les approches proposées.

4.6.1.1 Ant colony optimisation

Nous utilisons dans cette première solution, l'algorithme des colonies de Fourmies pour résoudre le DTP. Nous allons décrire dans ce qui suit l'adaptation des operateurs de cette métaheuristique au problème étudié.

1. **Les opérateurs de l'algorithme ACO**

   a) **Recherche locale**
   Dans la recherche locale que nous proposons, supposons que nous avons une solution. Nous parcourons le vecteur de la solution et on inverse la valeur des élément (si la valeur d'un élément est égale à 1, elle devient 0 ou inversement) . A chaque fois que nous inversons la valeur de chaque élément nous faisons une évaluation de la

fonction objectif. A tous coups où la solution est améliorée, cette dernière est sauvegardé et la recherche d'une meilleure solution se fera à partir de cette solution.

La recherche se terminera lorsque touts les nœuds dominants aurons été parcourus. il est à noter que la méthode de recherche locale citée ci dessus est utilisée dans toutes les solutions proposées.

b) **Les méthodes de sélection**
Nous proposons d'utiliser deux méthodes de sélection d'individus. Ces deux méthodes ont été retenues après une série d'expérimentations faisant apparaître les deux méthodes donnant les meilleurs résultats.

- **Méthode de sélection par Intensification :**
  Son principe consiste à choisir le nœud ayant la plus grande probabilité pour améliorer la solution
- **La sélection par tournoi**
  le principe de cette sélection consiste à choisir deux nœuds aléatoirement, puis de retenir celui ayant la meilleure probabilité

2. **Fonctionnement de l'algorithme ACO**
L'algorithme de colonie de fourmi est une approche métaheuristique basée sur une population et inspirée du comportement des fourmis réelles. L'idée principale derrière cette méthode provient de l'observation de la capacité des fourmis pour trouver le plus court chemin entre une source de nourriture et le nid. Cette méta-heuristique est composée d'une phase d'initialisation et d'une phase itérative. La première phase consiste à initialiser une solution de départ qui sera décrite ci-dessous. Elle comprend également l'initialisation du vecteur des probabilités et de la table de phéromone.

Ensuite, pour chaque itération, chaque fourmi construit sa solution en choisissant aléatoirement une méthode de sélection (intensification - sélection par tournois). Après avoir construit une solution la recherche locale est lancée pour atteindre un résultat meilleure. La meilleure solution retournée par la fourmi courante va participer à la mise à jour du vecteur de probabilité(mise à jour en ligne) en appliquant la formule (1). A la fin de chaque itération, la meilleure solution trouvée

par la population de fourmis va elle aussi participer à la mise à jour du vecteur de probabilité (mise à jour hors ligne) avec la formule suivante :

$$p_{ij}^{k} = \frac{[T_j]^{\alpha} [n_{ij}]^{\beta}}{\sum_{h \in S} \sum_{l \in N_h^k} [T_l]^{\alpha} [n_{hl}]^{\beta}} \tag{1}$$

où :
T_j la concentration de phéromone du nœud j
n_{ij} est le terme heuristique qui vaut $1/w_{ij}$, dont w_{ij} est le poids de l'arc du nœud i vers le nœud j.
$\alpha\,, \beta$ sont deux paramètres qui déterminent l'influence relative de la trace de phéromone et de l'information heuristique dans le processus de génération des solutions.
N_h^k est l'ensemble de sommets non sélectionnés adjacents à un sommet $h \in S$.

Nous allons décrire dans ce qui suit, l'approche de construction de la solution initiale. Soit S l'ensemble des solutions (initialement vide). Tous les sommets de notre graphe sont étiquetés initialement comme non marqués. Un sommet v est sélectionné aléatoirement dans l'ensemble V puis ajouté à S. Ce sommet est étiqueté comme marqué et tous les sommets adjacents à ce sommet deviennent marqués. Après cela, à chaque étape, la fourmi k construit une solution en sélectionnant un sommet non sélectionné v adjacent à un sommet déjà sélectionné $u \in S$. La sélection de ce sommet non sélectionné v est déterminée de manière probabiliste.

Après que tous les sommets du graphe soient étiquetés à marqué. Les fonction élagage et MST décrit dans le chapitre 3 sont appliquées.

Le pseudo-code de ACO est donné dans l'algorithme 1 suivant :

Algorithme 24 : ACO DT

Données : Un graphe général pondéré non orienté $G = (V, E, w)$
Résultat : Un DT optimal de G
Initialiser le vecteur de probabilité ainsi que la table de pheromone
Construction de la solution initiale
tant que *le nombre d'itération non atteint* **faire**
  pour *chaque fourmi* **faire**
    Construction d'une solution
    Recherche locale()
    Mise à jour de la table de phéromone. Selon la meilleure solution trouvée par la fourmi
  fin
  Mise à jour de la table de phéromone Selon la meilleur solution atteinte par les fourmis
fin
Retourner le DT avec poids minimum dans la liste

4.6.1.2 Algorithme génétique

Nous allons décrire dans ce qui suit les opérateurs retenus dans l'approche utilisant les algorithmes génétiques.

1. **Les opérateurs de l'algorithme ACO**

   a) **Génération de la Population initiale**
   La population initiale est construite avec un ensemble de solutions (individu) définies aléatoirement. La taille de chaque individu est le même nombre de nœuds dans un réseaux de capteurs sans fil.

   b) **Codage des éléments d'une population**
   Le codage utilisé dans cette approche est celui décrit dans la section II.

   c) **La méthode de croisement utilisée :**
   Pour cet algorithme nous avons utilisé la méthode de croisement uniforme. Il opère à l'aide d'un masque construit aléatoirement, pour décider lequel des parents va transmettre la valeur du gène à

l'un ou l'autre des descendants. Si à la même position que le gène, la valeur du masque est égale à 1, le gène du parent 1 passe à celui de l'enfant 1 et le gène du parent 2 passe à l'enfant 2. Sinon, c'est l'inverse qui se produit (figure 4.1). Cette fonction renvoie l'enfant ayant la meilleur qualité (le poids minimal).
La figure 4.1 illustre se processus par un exemple.

Parent 1	1	0	0	0	1	1	1	1	1	0
Parent 2	1	1	1	1	0	0	0	0	0	1
Masque	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**
Enfant 1	1	1	1	0	1	1	0	0	1	1
Enfant 2	1	0	0	1	0	0	1	1	0	0

Fig. 4.1 – Exemple de croisement uniforme

d) **Mutation**

La mutation est une autre solution pour créer de nouveaux individus et de modifier ceux déjà existant. Le hasard dans cette méthode va nous être d'une grande utilité. Dans notre cas, un index généré aléatoirement va décider dans quelle position le gène à muter se trouve. Rien ne nous dit que l'individu muté sera meilleure ou au moins bon, mais il pourrait être efficace pour la création de bonnes solutions. Un exemple illustratif sera représenté par la figure 4.2.

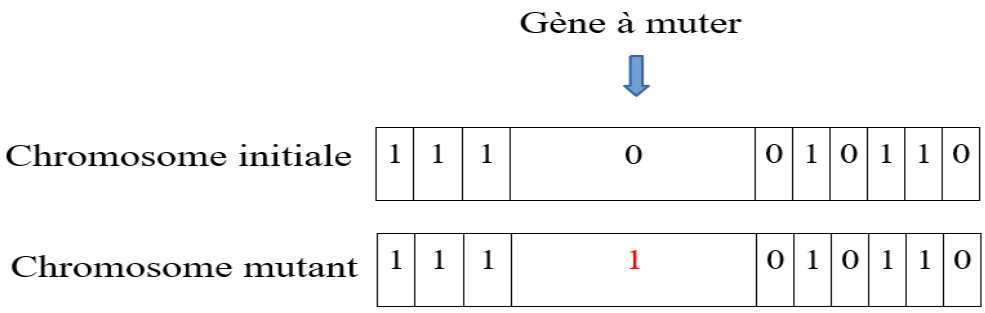

Fig. 4.2 – Représentation schématique de la mutation

e) **Élitisme**
Cet opérateurs permet de garder les meilleurs individus de chaque population. Notre méthode d'élitisme consiste à remplacer les m mauvais individus de la nouvelle population par les m meilleurs individus de la population précédente.

2. **Fonctionnement de l'algorithme génétique (GA)**

Après avoir créé une population initiale, l'algorithme GA procède comme suit :

Algorithme 25 : Génétique DT

```
Initialiser une population d'une façon aléatoire
tant que i < Nombre d'itération faire
    tant que j < PopSize faire
        Récupérer deux individu de la population n1, n2
        Enfant = Croisement(n1, n2)
        rand = Rand(0, 100)
        si rand < 5% alors
            Mutation(Enfant)
        fin
        Recherche locale()
        Ajouter le meilleur individu à la nouvelle population
    fin
    Elitisme()
fin
```

4.6.1.3 biogeography based optimimisation (BBO)

C'est une méta heuristique qui fait partie des algorithmes évolutionnaire basée sur la biogéographie. Dans ce qui suit, nous allons présenter l'approche BBO développée pour la résolution du problème de l'arbre dominant ainsi que ses opérateurs.

1. **Les opérateurs de l'algorithme BBO**
Il est à noter que le codage d'une solution et l'initialisation de la population sont les même que celles décrient pour l'algorithme génétique. BBO utilise principalement deux opérateurs : la migration et la mutation

a) **Migration**

Le concept générale de cet opérateur est de décider laquelle des régions de la solution migre ou non vers une autre région. La migration d'une région est le fait qu'un élément d'une solution s'interchange avec un autre élément d'une autre solution. Il est généralement utilisé pour modifier l'individu existant. L'algorithme 3 décrit le fonctionnement de l'opérateur de migration utilisé par BBO .

Algorithme 26 : Migration BBO DT

Données : position d'un individu donnée j
tant que $i <$ *taille de population* **faire**
  $rand = Random(0, 1)$
  si $rand < \mu(I_i)$ **alors**
    choisir un sommet de l'individu dans la position j aléatoirement et le permuter avec un autre sommet de l'individu dans la position i
  fin
fin

Où :
$\mu(I_i)$ Probabilité Emigration de l'individu I_i

b) **Mutation**

La mutation est un opérateur probabiliste. Le but de cet opérateur est de maintenir la diversité d'une population.

Algorithme 27 : Mutation BBO DT

Données : position d'un individu j, PM
tant que $i <$ *taille de l'individu* **faire**
  $rand = Random(0, 1)$
  si $rand < M_i$ **alors**
    permuter le nœud courant avec n'importe quel nœud définit dans une position aléatoire
  fin
fin

Où :
M_i est la probabilité de mutation.
$M_i = Pmut * (((1 - Pmut(j)))/PmutMax)$
dont : $Pmut$ est une constante qui définit la probabilité de muta-

tion.
$Pmut(j)$ est la probabilité de mutation de l'individu j.
$PmutMax$ est la probabilité de mutation maximale des individus de la population.

c) **Mise à jours des probabilité**
Les taux d'immigration (λ) et d'émigration (μ) de chaque individu I sont utilisés pour transmettre, de manière probabiliste, les caractéristiques entre les individus. Le taux d'immigration, quand il y a S individus dans la population, est donné pasr la formule (2) décrie si dessous :

$$\lambda(I_i) = 1 - \frac{nbv}{S} \tag{2}$$

Où :
nbv est le nombre d'individus qui ont une valeur de la fonction objectif pire que l'individu courant (I_i).

Le taux d'émigration quand il y a S individus dans la population est donné par :

$$\mu(I_i) = \frac{nbv}{S} \tag{3}$$

Quant aux taux de la mutation est calculé en utilisant les résultats des deux formules précédentes et est donné par :

$$Pmut(I_i) = \frac{\lambda_i/\mu_i}{\sum_{j=0}^{j=S} \lambda_j \,/\, \sum_{j=0}^{j=S} \mu_j} \tag{4}$$

2. **Le fonctionnement de l'algorithme BBO**
L'algorithme BBO que nous avons développé peut être décrit globalement par l'algorithme 28. Les deux opérateurs de base qui régissent le fonctionnement de BBO sont la migration et la mutation et sont les même décries si dessus. En plus, une stratégie d'élitisme est adoptée dans l'algorithme BBO, afin de garder dans la nouvelle population la

meilleure solution.

Algorithme 28 : Algoritme BBO DT

```
Initialisation de la population
tant que la condition d'arrêt non vérifiée faire
    Mise à jours des probabilité (Emigation, Immigration,
     Mutation)
    pour chaque Individu I_i de la population faire
        rand = Rand(0, 1)
        si rand < λ(I_i) alors
            Migration()
        sinon
            Mutation()
        fin
        Recherch Local()
    fin
    Elitisme ()
fin
```

4.6.2 Description de l'approche coopérative

Nous proposons une approche coopérative entre les méta-heuristiques pour la résolution du problème de l'arbre dominant où une méta-heuristique maître contrôle plusieurs méta-heuristiques esclaves. Les points forts d'une méta-heuristique esclave compensent les points faibles d'une autre grâce à la notion de rang. Le rang est un moyen de mettre en valeur les points forts des méta-heuristiques esclaves. Il est attribué par le maître et est égale à 0 au départ. Il est mis à jour continûment par la méta-heuristique maître qui augmente le rang des méta-heuristiques esclaves ayant amélioré la solution et diminue le rang de celles qui ne l'auront pas amélioré. Cette mise à jour tend à récompenser les méta-heuristiques esclaves qui auront réussi à améliorer la solution courante. Comme conséquence, les méta-heuristiques esclaves ayant un rang élevé participeront donc davantage au développement de la solution finale.

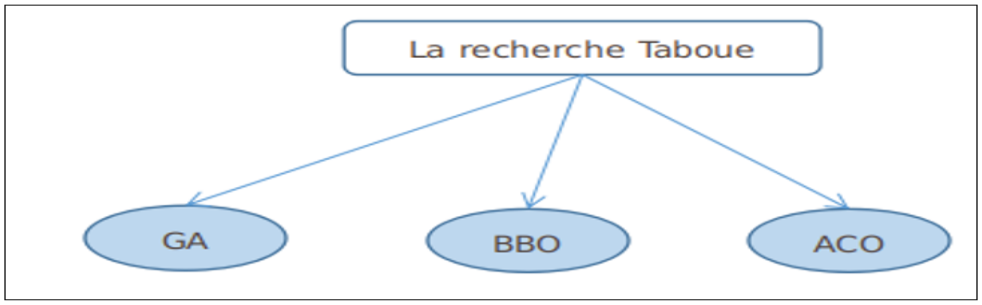

Fig. 4.3 – Coopération des méta-heuristiques

Comme le montre la figure 4.3, nous utilisons comme méta-heuristique maître la recherche taboue, et pour les méta-heuristiques esclaves nous utilisons l'algorithme génétique, ACO ainsi que BBO. Au départ, le maître choisit une méta-heuristique esclave de manière aléatoire. A chaque fois qu' une méta-heuristique esclave améliore sa solution, elle aura comme bonus une valeur supérieure de rang, cela crée une forme de compétition entre les esclaves pour accroître son rang tout en améliorant la solution actuelle.

Lorsqu'une méta-heuristique termine sa recherche, elle communique sa meilleure solution à celle qui la remplace, permettant ainsi à cette dernière de générer un bon voisinage. La sélection de la méta-heuristique remplaçante se fait par rapport à la valeur du rang (c'est-à-dire la méta-heuristique ayant le plus grand rang sera choisie pour l'amélioration de la solution courante). La méta-heuristique esclaves n'ayant pas amélioré la solution après un nombre d'itération précis elle sera mis dans la liste taboue. Lorsque tous les méta heuristique sont dans la lites tabou la méta heuristique maître libère l'esclave ayant un rand élevé pour contribué à la recherche d'une meilleure solution. L'algorithme 29 présente le processus

de fonctionnement de la méta-heuristique maître :

Algorithme 29 : Algorithme de la méta-heuristique maître

```
tant que condition d'arrêt non vérifiée faire
    Choisir la méta-heuristique ayant uen rand élevé et qui n'est pas
     dans la liste taboue
    si la méta-heuristique choisi donne une meilleure fitness alors
        rand = rand + α
        Vider la liste taboue
    sinon
        rand = rand + α
        Mettre la méta-heuristique esclave courante dans la liste
         taboue
    fin
fin
```

Où :
α représente : une valeur positive, si la qualité de la solution est améliorée, sinon α sera négatif.

L'algorithme suivant va définir le processus générale de la méthode de coopération

Algorithme 30 : Algorithme de coopération

```
tant que condition d'arrêt non vérifiée faire
    si la list taboue est pleine alors
        récupérer la méta-heuristique ayant un rand elevé
    sinon
        choisir la méta-heuristique ayant un rand élevé et qui n'est
         pas dans la liste taboue
    fin
    executer la méta choisie et récuperer la solution trouvé S
    si S est meilleure que la meilleure solution trouvée SB alors
        SB = S
        Vider la liste taboue
    sinon
        mettre la méta dans la lise taboue
    fin
    mise à jour du rand
fin
```

4.7 Conclusion

A travers ce chapitre, nous avons présenté de manière détaillée les méta-heuristiques que nous avons proposé pour la résolution du problème de l'arbre dominant, ainsi que les différentes méthodes et opérateurs contribuant à leurs élaborations. Nous avant aussi proposé une nouvelle approche qui consiste à faire coopérer plusieurs méta-heuristique (esclaves) pour travailler dans l'espace des solutions supervisés par la méta-heuristique maître qui travaille dans l'espace des méta-heuristiques. Dans le but d'évaluer la qualité de ces approches, le prochain chapitre portera sur les différents résultats obtenus à travers les expérimentations effectuées.

Bibliographie

[1] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 64. IEEE Press, 2005.

[2] Holger Karl and Andreas Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[3] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[4] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12) :2292–2330, 2008.

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

[6] Ning Zhang, Incheol Shin, Bo Li, Cem Boyaci, Ravi Tiwari, and My T Thai. New approximation for minimum-weight routing backbone in wireless sensor network. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 96–108. Springer, 2008.

[7] Mohemed Mekidiche and Hichem Rais. *Etude et évaluation du protocole de localisation DV-HOP dans un réseau de capteurs sans fil.* PhD thesis, http ://dspace.univ-tlemcen.dz/handle/112/7105, 2015.

[8] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ACM SIGOPS operating systems review*, volume 34, pages 93–104. ACM, 2000.

[9] Jessica Feng, Farinaz Koushanfar, and Miodrag Potkonjak. System-architectures for sensor networks issues, alternatives, and directions. In *null*, page 226. IEEE, 2002.

[10] Abdelmalik Bachir, Mischa Dohler, Thomas Watteyne, and Kin K Leung. Mac essentials for wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 12(2) :222–248, 2010.

[11] GUNN Meghan and GM Simon. A comparative study of medium access control protocols for wireless sensor networks. *International Journal of Communications, Network and System Sciences*, 2(08) : 695, 2009.

[12] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38 (4) :393–422, 2002.

[13] Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4) :921–960, 2007.

[14] Tim Nieberg, Stefan Dulman, Paul Havinga, Lodewijk van Hoesel, and Jian Wu. Collaborative algorithms for communication in wireless sensor networks. In *Ambient Intelligence : Impact on Embedded Sytem Design*, pages 271–294. Springer, 2003.

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

[16] Myung Ah Park, James Willson, Chen Wang, My Thai, 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.

[17] My T Thai, Ravi Tiwari, and Ding-Zhu Du. On construction of virtual backbone in wireless ad hoc networks with unidirectional links. *IEEE Transactions on Mobile Computing*, 7(9) :1098–1109, 2008.

[18] 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.

[19] Shyam Sundar. A steady-state genetic algorithm for the dominating tree problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 48–57. Springer, 2014.

[20] Kamel Zidi. *Système interactif d'aide au déplacement multimodal (SIADM)*. PhD thesis, Ecole Centrale de Lille ; Université des Sciences et Technologie de Lille-Lille I, 2006.

[21] 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.

[22] Jakob Puchinger and Günther R Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53. Springer, 2005.

[23] Alexander Schrijver. Theory of linear and integer programming. johnwiley & sons. *New York*, 1986.

[24] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1) :60–100, 1991.

[25] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1) :1–7, 1987.

[26] Mark Zlochin, Mauro Birattari, Nicolas Meuleau, and Marco Dorigo. Model-based search for combinatorial optimization : A critical survey. *Annals of Operations Research*, 131(1-4) :373–395, 2004.

[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] Ernesto Bonomi and Jean-Luc Lutton. The n-city travelling salesman problem : Statistical mechanics and the metropolis algorithm. *SIAM review*, 26(4) :551–568, 1984.

[29] René VV Vidal. *Applied simulated annealing*, volume 396. Springer, 1993.

[30] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549, 1986.

[31] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2) :126–140, 1994.

[32] Alain Hertz and Daniel Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126(1) :1–12, 2000.

[33] JH Holland. Adaptation in natural and artificial systems : an introductory analysis. *Holland, JH*, 1975.

[34] Ingo Rechenberg. Evolutionsstrategie : Optimierung technischer systeme nach prinzipien der biologischen evolution, frommann–holzboog. *Stuttgart, Germany*, 1973.

[35] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

[36] John R Koza. Genetic programming : On the programming of computers by means of natural selection (complex adaptive systems). *A Bradford Book*, 1 :18, 1992.

[37] Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1) :73–91, 1999.