**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.

**I. 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.

**II. 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.

**III. 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.

**IV. La méthode de connection:**

Soit un vecteur DNC initialement vide qui va contenir par la suite ensemble de nœuds connectés. 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 dans le cas d’un seul nœud à selectionner. Nous ré-exécutons ce prceossus jusqu'à ce que DN devient null.

Une fois que l’ensemble DN soit null, nous appliquons la méthode d’élagage et MST décritent 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).

**IV. 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.

**partie 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.

**Première solution proposée: 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.

**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 sauvegardée et la recherche d’une meilleure solution se fera à partir de cette solution.

La recherche se terminera lorsque tous 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:

(1)

**où:**

**Ʈj** la concentration de phéromone du nœud j

ƞij est le terme heuristique qui vaut 1 / wij, dont wij est le poids de l’arc du nœud i vers le nœud j.

α, β 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.

est l'ensemble de sommets non sélectionnés adjacents à un sommet h ∈ 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∈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 1**: ACO de DT

1. **Entrées**: un graphe général pondéré non orienté G=(V, E, w)

2. **Sortie**: Un DT optimal de G

3. Initialiser le vecteur de probabilité ainsi que la table de pheromone.

4. Construction de la solution initiale.

5. **Tantque** le nombre d’itération non atteint **faire**

6. **pour** chaque fourmi **faire**

7. Construction d’une solution

8. Recherche locale()

9. Mise à jour de la table de phéromone. Selon la meilleure solution trouvée par la fourmi

10.  **Fin pour**

11. Mise à jour de la table de phéromone.

Selon la meilleur solution atteinte par les fourmis

12. **Fin Tant que**

13. Retourner le DT avec poids minimum dans la liste

Deuxième solution proposée: Algorithme génétique

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

**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 1). Cette fonction renvoie l’enfant ayant la meilleur qualité (le poids minimal). La figure 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 |

**Figure 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 2.

Gène à muter

Chromosome initiale

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

Chromosome mutant

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

**Figure 3:** 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. Le fonctionnement de l’algorithme génétique (GA)**

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

**Algorithme 2**: Génétique

1. Initialiser une population d’une façon aléatoire.

2. **Tant que** ( i<Nombre d’itération)

3. **Tant que** ( j<PopSize)

4. Récupérer deux individu de la population n1,n2

5. Enfant= Croisement (n1,n2)

6. rand=Générer une valeur aléatoire

7. **Si** ( rand<5) //la probabilité d’avoir une mutation est 5%

8. Mutation(Enfant)

9. **FinSi**

10. Recherche locale()

11. Ajouter le meilleur individu à la nouvelle population.

12. **Fin Tant que**

13. Elitisme()

14. **Fin Tant que**

**Troisième solution proposée: 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 3**: Migration

1. **Entrées**: position d’un individu donnée j

2. **Tant que** ( i< taille de population)

3. val= get random val() // récupérer une valeur aléatoirement

4. **Si**(val < Probabilité Emigration(i))

5. 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

6. **Fin tant que**

**b. Mutation**

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

**Algorithme 4**: Mutation

1. **Entrées**: position d’un individu j, PM

2. **Tant que** ( i< taille de l’individu ) **faire**

3. val= get random val() // récupérer une valeur aléatoirement

4. if(val <mi)

5. permuter le nœud courant avec n’importe quel nœud définit dans une position aléatoire

6. **Fin tant que**

où:

dont: Pm est une constante qui définit la probabilité de mutation.

Pmut(j) est la probabilité de mutation de l’individu j.

PmutMax: est la probabilité de mutation maximale des individus de la population.

Mi est la probabilité de mutation.

**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:

(2)

où nbv est le nombre d’individus qui ont une valeur de la fonction objectif pire que l’individu courant (Ii).

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

(3)

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

(4)

**2. Le fonctionnement de l’algorithme BBO**

L’algorithme BBO que nous avons développé peut être décrit globalement par l’algorithme 5. 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 5**: Algoritme BBO

1. Initialisation de la population

2. **Tant que** (la condition d’arrêt non vérifiée) **faire**

3. Mise à jours des probabilité (Emigation, Immigration, Mutation)

4. **Pour chaque** habitat Hi **faire**

5. **Si** proba d’Immigration > seuil définie aléatoirement **alors**

6. Migration()

7. **Fin si**

8. Mutation()

9. Recherche locale()

11. **Fin Pour**

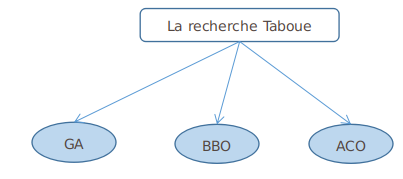
10. Evaluation de la population

11. Elitisme ()

13. **Fin tant que**

**Partie 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.



**Figure 4:** Coopération des méta-heuristiques.

Comme le montre la figure 4, 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 toutes les méta-heuristiques sont dans la listes tabou la méta-heuristique maître libère l’esclave ayant un rand élevé pour contribuer à la recherche d’une meilleure solution. L'algorithme 7 présente le processus de fonctionnement de la méta-heuristique maître :

**Algorithme 7**: Algorithme de la méta-heuristique maître

1.**Tant que** condition d’arrêt non vérifiée **faire**

2. Choisir la méta-heuristique ayant un rand élevé et qui n’est pas dans la liste taboue

3. **si** (fitness.actuelle<fitness)

4. rand = rand+α

5. Vider la liste taboue

4. **sinon** rand = rand+ α

6. Mettre la méta-heuristique esclave courante dans la liste taboue

7. **Fin si**

8. **Fin tant que**

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 8**: Algorithme de coopération

1.**Tant que** condition d’arrêt non vérifiée **faire**

2. Si la liste taboue est pleine

3. Récupérer la méta-heuristique ayant un rand elevé

4. Sinon choisir la méta-heuristique ayant un rand élevé et qui n’est pas dans la liste taboue

5. Fin si

6. Executer la méta choisie et récuperer la solution trouvé S

7. Si S est meilleure que la meilleure solution trouvée SB

8. SB=S

9. Vider la liste taboue

10. Sinon mettre la méta dans la lise taboue

11. fin si

12. mise à jour du rand

13. fin tant que

**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.