République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A. Mira de Béjaia

Faculté des Sciences Exactes

Département de Recherche Opérationnelle

MÉMOIRE DE FIN DE CYCLE

en vue de l'obtention du diplôme de Master en Recherche Opérationnelle

Option : Modélisation Mathématiques et Techniques de Décisions

Thème

Résolution de jeux par les métaheuristiques

Présenté par :

✓ M^{lle}. **AMARA** Kahina.

✓ M^{r}. **MEHENNI** Massinissa.

Devant le jury composé de :

Encadreur	M^{lle}. BOUCHAMA Kahina	MAA	U. A. Mira Béjaia.
Présidente	M^{lle}. AOUDIA Zohra	MAA	U. A. Mira Béjaia.
Examinateur	M^{r}. TOUATI Sofiane	MAA	U. A. Mira Béjaia.
Examinateur	M^{r}. ZIANI Sofiane	Docteur	U. A. Mira Béjaia.

Béjaïa 2017

Remerciements

Nous remercions Dieu tout puissant de nous avoir accordé santé, courage et la volonté pour accomplir ce travail.

Nous tenons également à remercier notre encadreur M^{lle} K.BOUCHAMA pour l'aide et l'assistance qu'elle nous a fournit afin de nous permettre de mener à bien et à terme ce mémoire de fin d'études, et qu'il nous soit permis de lui exprimer notre profonde gratitude.

Nous exprimons notre grand respect aux honorables membres du jury qui ont accepté d'évaluer ce travail.

Nous tenons tout simplement à exprimer notre profonde gratitude à tous ceux qui nous ont soutenus de prés ou de loin durant tout notre cursus.

K.AMARA - M.MEHENNI

Dédicaces

***J**e dédie ce travail à :*

***M**es parents, les deux personnes qui comptent le plus pour moi.*

***S**alim, Yacine, Zakia & Zina, les plus merveilleux des frères et sœurs.*

***M**es chers grands parents.*

***M**es oncles et mes tantes.*

***M**es adorables tata, Zakia & Salima.*

***M**eriem & Nissa, les deux amies et sœurs les plus chères à mon cœur.*

***C**hafia, Hanane, Nassima, Nédhal, Sarah, Sofiane, Sofiane, Walid & Zahra, les meilleurs amis du monde.*

***M**on binôme Massinissa avec qui je partage ce travail.*

A.Kahina

Dédicaces

***J**e dédie ce modeste travail à :*

***M**es parents.*

***M**es frères & sœurs.*

***M**es oncles et mes tantes.*

***M**a binôme Kahina avec qui je partage ce travail.*

***T**ous mes amis.*

M.Massy

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

Le général qui gagne une bataille a médité, calculé, avant de combattre, le général qui perd une bataille a fait moins de calculs. Donc, beaucoup de stratégies mènent à la victoire, peu de stratégies à la défaite : que dire du hasard total ?

- Sun Tzu -

Quotidiennement, des agents tels que des entreprises, des individus, des économistes, etc, sont confrontés à des situations d'interactions stratégiques où le sort de chacun dépend non seulement de ses propres décisions, mais aussi de celles prises par les autres. Pour étudier ces situations de manière rationnelle, *Von Neumann, Morgenstern (1944) et John F.Nash (1951)* ont contribué à l'élaboration d'une discipline théorique dite **Théorie des jeux**, qui permet de comprendre formellement ces situations dans lesquelles les preneurs de décisions (joueurs), interagissent. Un jeu est alors défini comme un univers dans lequel chaque preneur de décision possède un ensemble d'actions possibles déterminé par les règles du jeu.

Il est convenu de distinguer deux grandes familles de jeux, qu'on qualifie de coopératif si les joueurs sont en mesure de passer des accords qui les lient de manière contraignante et mener conjointement leurs décisions vers une situation qui leur soit collectivement profitable, ou non coopératif lorsque ceux-ci sont libres dans leurs choix et poursuivent des objectifs propres et indépendants.
Développer de tels modèles théoriques, va de soi pour des concepts de solutions. Parmi les différents concepts existants, nous citons l'*équilibre de Nash*, qui est fondamental en théorie des jeux. C'est une situation dans laquelle aucun joueur n'a intérêt à dévier

unilatéralement de sa stratégie tant que ses adversaires maintiennent les leurs.

L'existence de cet équilibre en stratégies mixtes pour un jeu fini a été prouvée par celui à qui il doit son nom. Depuis, plusieurs algorithmes ont été proposés, mais la question qui se pose, ces algorithmes ; peuvent-ils trouver cet équilibre en un temps raisonnable ? *Chritos Papadimitriou* a prouvé que le calcul de l'équlibre de Nash pour un jeu à N-joueurs, appartient à la classe $PPAD$ qui est une sous classe de la classe NP.
Il est bien connu que calculer l'équilibre de Nash en stratégies pures est un problème complexe. Cependant, bien que pas nombreux, il existe certains travaux contemporains dans la littérature qui ont étudié les métaheuristiques pour le calcul de cet équilibre.

L'objectif assigné à ce mémoire, consiste dans un premier temps à modéliser le problème de clustering de données en un problème de jeux non coopératifs. Par la suite, adapter un algorithme génétique basé sur un processus itératif de recherche des stratégies de meilleures réponses, convergeant vers un équilibre de Nash, que nous détaillerons beaucoup plus dans le dernier chapitre.

Nous avons réparti ce mémoire en quatre chapitres, le premier chapitre comporte les rappels et concepts de base de l'optimisation combinatoire ainsi que ses problèmes fondamentaux et ses méthodes de résolution dont les métaheuristiques les plus répandues.

Puis, le second chapitre sera consacré aux notions fondamentales de la théorie des jeux, qui nous donnera un aperçu global sur cette dernière.

Dans le troisième chapitre, nous présenterons une synthèse des travaux sur les différentes approches de résolution de jeux par les métaheuristiques.

Et enfin, dans le quatrième et dernier chapitre, nous exposerons notre modèle de jeu de clustering de données ainsi que son approche de résolution que nous avons élaboré à l'aide des algorithmes génétiques.
Nous terminons ce travail avec une conclusion et perspectives.

CHAPITRE 1

OPTIMISATION COMBINATOIRE ET MÉTAHEURISTIQUES

Introduction

L'objectif principal de l'optimisation combinatoire est de trouver une meilleure solution parmi un nombre fini (souvent très grand) de choix. C'est une branche de l'optimisation, appelée aussi 'optimisation discrète', qui recouvre des méthodes servant à déterminer l'optimum d'une (ou plusieurs) fonctions, sous certaines contraintes.
Dans ce chapitre, nous allons décrire brièvement quelques notions de l'optimisation combinatoire, ses problèmes classiques, ainsi que quelques méthodes de résolutions en particulier les métaheuristiques.

1.1 Définition d'un problème d'optimisation combinatoire

En mathématique, un problème d'optimisation consiste à chercher une instanciation d'un ensemble de variables qui peuvent être soumises (ou pas) à des contraintes, de façon à maximiser ou minimiser un ou plusieurs critères. Lorsque le domaine de valeurs des variables est discret, on parle alors de problème d'optimisation combinatoire (POC).
Formellement un problème d'optimisation est défini comme suite :

$$(P) : \begin{cases} Min\ f(s) \\ s \in S \end{cases} \tag{1.1}$$

où ;
S : l'ensemble des solutions réalisables ou admissibles.
$f : S \to \mathbb{R}$ la fonction objectif à minimiser.

Le problème de minimisation (P) se ramène facilement à un problème de maximisation telle que :

$$\begin{cases} Max\ f(s) = -Min(-f(s)) \\ s \in S \end{cases} \tag{1.2}$$

Pour un problème de minimisation, il s'agit de trouver une solution $s \in S$ réalisable, qui minimise la fonction f.
Une solution $s \in S$ pour le problème (1.1) peut être :

- **Minimum local :** Une solution $s \in S$ est un minimum local pour la fonction objectif f s'il existe un voisinage $N(s)$, tel que $\forall s' \in N(s), f(s) \leqslant f(s')$ [41].
- **Minimum global :** Une solution $s \in S$ est un minimum global pour la fonction objectif f si $\forall s' \in S, f(s) \leqslant f(s')$ [41].

1.2 Quelques problèmes classiques en optimisation combinatoire

1.2.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce, étudié depuis le 19^{me} siècle, est l'un des plus anciens problèmes d'optimisation combinatoire. Il s'agit d'un représentant de commerce ayant n villes à visiter, qui souhaite établir une tournée en passant exactement une fois par chaque ville puis revenir à son point de départ pour un moindre coût, c'est-à-dire en parcourant la plus petite distance possible. La notion de distance peut être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense pour effectuer la tache [42] [51].

Un tel problème peut se formuler de la manière suivante :

$$\begin{cases} Min\ F = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\ sc \quad \sum_{j=1}^{n} x_{ij} = 1 & \forall i = \overline{1,n} & \\ \quad \sum_{i=1}^{n} x_{ij} = 1 & \forall j = \overline{1,n} & \\ \quad \sum_{i\in S}\sum_{j\in \overline{S}} x_{ij} \geq 1 & \forall S \subset X/\ 2 \leq |S| \leq n-1 & \\ \quad x_{ij} \in \{0,1\} & \forall i = \overline{1,n}, & \forall j = \overline{1,n} \end{cases} \tag{1.3}$$

Où :

$$x_{ij} = \begin{cases} 1, \text{ si la ville } j \text{ suit immédiatement la ville } i \text{ dans le parcours} \\ 0, \text{ sinon.} \end{cases} \quad \forall i = \overline{1,n},\ \forall j = \overline{1,n}$$

c_{ij} : est la distance entre la ville i et la ville j.

X : est l'ensemble des villes.

1.2.2 Le problème du sac à dos

Le problème du sac à dos fait partie des problèmes d'optimisation combinatoire les plus étudiés en raison de ces nombreuses applications dans le monde réel. Il s'agit d'un problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource.

Étant donné un sac à dos de capacité maximale b, et n objets possédant chacun un poids et une valeur, le problème qui se pose est le suivant : comment remplir le sac à dos de sorte que le poids total des objets choisis n'excède pas sa capacité b, tout en maximisant la valeur totale des objets [40] [42].

Ce problème se formule de la manière suivante :

$$\begin{cases} Max\ F = \sum_{j=1}^{n} p_j x_j \\ sc \quad \sum_{j=1}^{n} \omega_j x_j \leqslant b \\ x_j \in \{0,1\}, j = \overline{1,n} \end{cases} \tag{1.4}$$

Où :

$$x_j = \begin{cases} 1, \text{ si l'objet } j \text{ est sélectionné} \\ 0, \text{ sinon.} \end{cases} \qquad \forall j = \overline{1,n}$$

b : La capacité du sac.

ω_j : Le poids de l'objet j.

p_j : La valeur de l'objet j.

1.2.3 Le problème d'affectation

Le problème d'affectation consiste à établir des liens entre les éléments de deux ensembles distincts, de façon à minimiser un coût tout en respectant les contraintes d'unicité de lien pour chaque élément.

Étant donnés m tâches et n ouvriers, l'affectation de la tâche i à l'ouvrier j entraîne un coût de réalisation noté c_{ij}.

Cette dernière se fait de sorte que :

- Chaque ouvrier j ait une seule tâche.
- Chaque tâche i est attribuée à un seul ouvrier.

Donc, le problème consiste à trouver une affectation de coût minimum [14] [42].
Ce problème se formule de la manière suivante :

$$\begin{cases} Min\ F = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \quad sc \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = \overline{1,m} \\ \qquad\quad \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = \overline{1,n} \\ \qquad\quad x_{ij} \in \{0,1\} \quad \forall i = \overline{1,m},\ \ \forall j = \overline{1,n} \end{cases} \tag{1.5}$$

Où :

$$x_{ij} = \begin{cases} 1, \text{ si la tâche } i \text{ est affectée à l'ouvrier } j \\ 0, \text{ sinon.} \end{cases} \quad \forall i = \overline{1,m},\ \ \forall j = \overline{1,n}$$

c_{ij} : est le coût de l'affectation de la tâche i à un ouvrier j.

1.2.4 Le problème d'ordonnancement

Le problème d'ordonnancement consiste à séquencer et à placer dans le temps, un ensemble d'activités (entités élémentaires de travail), compte tenu de contraintes temporelles (délais, contraintes d'enchaînements, ...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les activités, pour optimiser un ou plusieurs objectifs (délai total de réalisation du projet, coût total du projet, ...) [4] [25].

Il s'agit d'un problème de satisfaction de contrainte qui trouve ses applications dans divers domaines tels que la gestion de projets, ateliers de production, ...
Contrairement aux autres problèmes que nous avons définis précédemment, ce problème ne fait pas référence à un problème totalement défini pour lequel il existe une formulation mathématique directe, mais plutôt à une famille de problèmes. En effet, un problème d'ordonnancement est défini par la donnée des activités et des ressources qui le constituent, ces éléments peuvent être de nature très variée [42] [50].

1.3 Complexité algorithmique

Avant d'aborder les différentes méthodes de résolution des problèmes d'OC, nous introduisons quelques définitions et notions sur l'analyse de leur complexité.
Cette dernière a pour objectif de quantifier le temps d'exécution d'un algorithme, ainsi que la taille mémoire nécessaire pour contenir et manipuler les programmes et leurs structures de données, dans le but de comparer différentes techniques de résolution algorithmiques d'un problème donné.
Cette phase d'analyse s'impose uniquement pour des problèmes où l'espace de solutions est très grand.

1.3.1 Notions sur la complexité

La complexité d'un algorithme est un concept fondamental qui permet de mesurer sa performance asymptotique dans le pire et le meilleur des cas.
En terme de temps, la complexité d'un algorithme détermine son efficacité, elle est exprimée par une fonction $f(n)$ de $\mathbb{N}$ dans $\mathbb{R}$ qui représente le nombre d'opérations élémentaires qu'il effectue, où n est la taille des données à traiter.
Il ne s'agit pas de faire un décompte exact du nombre d'opérations $f(n)$ mais plutôt de donner un ordre de grandeur de ce nombre représenté par la notion de *Landau* $O(.)$

Définition 1.1. Soit $f(n)$ et $g(n)$ deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$. On dit que $f(n)$ est en $O(g(n))$ si :

$$\exists\ c \in \mathbb{R}^{+*}, \exists\ n_0 \in \mathbb{N}\ \text{, tels que } \forall\ n > n_0, f(n) \leq c * g(n)$$

Cette notation indique que dans le pire ou le meilleur des cas, la croissance de $f(n)$ ne dépassera pas celle de la fonction $g(n)$ [8].

1.3.2 Classes de complexité

On définit plus communément ces classes par les ordres de grandeurs suivants :
- Les algorithmes logarithmiques $O(log\ n)$, sont considérés comme étant excellents.
- Les algorithmes linéaires $O(n)$, sont considérés comme étant rapides.

– Les algorithmes polynomiaux $O(n^k)$, sont considérés lents pour $k > 3$.
– Les algorithmes exponentiels $O(2^n)$, sont considérés impraticables dès que la taille des données devient plus grande(supérieurs à quelques dizaines d'unités).

1.3.3 Classes de problèmes

La complexité des algorithmes a abouti à une classification des problèmes en fonction des performances de meilleurs algorithmes connus pour leur résolution.
On distingue quatre classes de problèmes :

Classe P : La classe des problèmes P est considérée comme étant la plus facile en terme de difficulté de ses problèmes, c'est à dire qu'il existe un algorithme qui consomme un temps polynomial pour résoudre un problème appartenant à cette classe [14].

Classe NP : On qualifie de NP, les problèmes de décision Π [1], qui peuvent être résolus par un algorithme non déterministe dans un temps polynomial [14].

Classe NP-Complet : Ce sont les problèmes de NP les plus difficiles. On dit qu'un problème de décision Π appartenant à NP est NP-Complet si on peut mettre en évidence une réduction polynomiale [2] de tout problème Π' dans NP à Π [16].

Classe NP-Difficile : On qualifie les problèmes d'optimisation Γ (ceux dont la réponse est de type numérique) comme étant NP-Difficiles. À un problème d'optimisation NP-Difficile on peut associer un problème de décision NP-Complet dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif est supérieure (respectivement inférieure) ou égale à une valeur donnée. Pour prouver qu'un problème d'optimisation Γ est NP-Difficile, il suffit de montrer que le problème de décision Π associé à Γ est NP-

1. Un problème de décision est un problème dont la réponse est de type binaire (oui/non, vrai/faux/, 0/1) [16].

2. Un problème de décision Π se réduit polynomialement à un problème de décision Π' s'il existe un algorithme résolvant Π, qui fait appel à un algorithme résolvant Π', et si l'algorithme de résolution de Π est polynomial lorsque cet appel est comptabilisé comme une opération élémentaire [20].

Complet. Il est à noter que jusqu'à maintenant, aucun algorithme polynomial n"est connu pour résoudre ce type de problèmes [16] [29] .

1.4 Approches de résolution

Le choix d'une méthode de résolution d'un problème d'optimisation n'est pas évident, il dépend d'un ensemble de facteurs liés au problème à optimiser, particulièrement à sa complexité. Pour faire le choix adéquat de la méthode, il est nécessaire d'avoir une bonne connaissance des outils d'optimisation, constitués d'un ensemble de méthodes qui se répartissent en deux grande catégories : les méthodes exactes et les méthodes approchées (heuristiques et méta-heuristiques).

1.4.1 Les méthodes exactes

Ces méthodes explorent de façon systématique l'espace des solutions jusqu'à trouver une solution optimale au problème traité. Elles garantissent l'optimalité mais pas l'efficacité (en terme de temps de calcul), car certaines instances de problèmes ne peuvent être résolues en un temps acceptable par ces approches exhaustives. Parmi les nombreuses méthodes exactes qui permettent de résoudre certain problèmes en un temps fini on cite comme exemple : les méthodes de la théorie des graphes (Bellman, Ford Fulkerson...), les méthodes de recherche arborescente (Branch and Bound), la programmation dynamique, etc [16] [51].

1.4.2 Les méthodes approchées

1.4.2.1 Les heuristiques

En optimisation combinatoire, une heuristique est une méthode approchée qui a pour but de trouver une solution réalisable en un temps raisonnable, pas nécessairement optimale. Généralement, une heuristique est conçue pour un problème particulier en s'appuyant sur sa propre structure [20].
L'usage d'une heuristique est efficace pour accélérer le processus de résolution exacte. Elle peut être appliquée à n'importe quelle classe de problèmes, qu'ils soient faciles ou difficiles.

1.4.2.2 Les métaheuritiques

Contrairement aux heuristiques qui construisent une ou plusieurs solutions en partant d'une solution vide, les métaheuristiques débutent par une solution initiale arbitraire $s_0 \in X$ sur l'ensemble des solutions réalisables et tentent de rendre meilleure celle-ci.
Ces méthodes sont prisées pour leur efficacité et leur flexibilité en termes d'adaptation à n'importe quel problème d'optimisation. Le point fort de ces méthodes est leur aptitude à échapper aux minima (respectivement maxima) **locaux**, en acceptant par fois durant l'exécution de leur programmes, des solutions qui dégradent temporairement la fonction objectif afin d'explorer toute solution potentielle qui pourrait être un minimum (respectivement maximum) **global**.
Nous citons quelques méthodes, les plus répandues :

- Méthodes de recherche locale (à solution unique)
Ces méthodes partent d'une solution initiale (obtenue par tirage aléatoire, ou provient d'une autre méthode approchée) et la déplace de façon itérative vers une solution voisine. Cette méthode est applicable seulement si, une notion de voisinage est définie.
Habituellement, chaque solution a plus d'une solution voisine ; le choix de celle vers laquelle se déplacer est pris en utilisant seulement les informations sur les solutions voisines de la solution courante, d'où le terme de **recherche locale**. Ces méthodes ne manipulent qu'une seule solution à la fois [3] [27].
Nous citons entre autre :

Le recuit simulé

Le recuit simulé est une mataheuristique inspirée d'un mécanisme naturel en métallurgie, celui de l'alliage d'un élément métallique avec d'autres éléments chimiques. Il se base sur les travaux faits par *Metropolis* en 1953, qui décrivent le processus de recuit physique consistant à chauffer un matériau à une température suffisante pour que les atomes retrouvent la liberté du déplacement afin d'aboutir à un équilibre physico-chimique. Cette opération est suivie d'un refroidissement lent où les atomes cherchent à former de nouvelles liaisons, pour obtenir une structure régulière du matériau à l'état solide.
Un état donné de la matière représente dans la méthode du recuit simulé, une solution s

du problème à résoudre, l'énergie du système représente la fonction objectif f à optimiser, donc un état d'équilibre où l'énergie $f(s)$ est optimale représente la solution optimale du problème [5] [16].

Les étapes de l'algorithme du recuit simulé sont représentées comme suit :

Algorithme 1.1 - Recuit Simulé [16] [20]

1. Générer aléatoirement une configuration initiale $s = s_0$ dont correspond l'énergie initiale $E = E_0$,
2. Initialiser la température $T = T_0$ en fonction du schéma de refroidissement,
3. Générer d'une manière aléatoire une configuration voisine s' de la configuration actuelle s, en déplaçant au hasard un atome quelconque,
4. Calculer la variation de l'énergie $\Delta E = f(s') - f(s)$
   Si $\Delta E < 0$, alors la nouvelle solution améliore la fonction objectif et diminue l'énergie, donc elle est acceptée.
   Sinon, elle sera acceptée avec une probabilité égale à $e^{\frac{-\Delta E}{T}}$
5. Répéter 3. et 4. jusqu'à ce que la configuration optimale soit atteinte.
6. Décroitre la température et répéter jusqu'à ce que le système se solidifie.

La Recherche Tabou

La recherche tabou est une méthode de recherche locale introduite dans les années 80 par *Fred Glover*. Elle est basée sur des mécanismes inspirés de la mémoire humaine. Elle se distingue des méthodes de recherche locale simples par l'utilisation d'une mémoire appelée liste tabou de taille k, qui enregistre les k dernières solutions visitées vers lesquelles il est interdit de se déplacer (d'où le nom attribué à la méthode par Glover). Cette liste est utilisée lors de déplacements dans le voisinage dans le but de favoriser une large exploration de l'espace des solutions et d'éviter d'y retourner trop rapidement à des solutions déjà visitées.

En effet, à partir d'une configuration courante s, on choisit une solution $s' \in N(s)$ en dehors des éléments de cette liste tabou T, même si elle dégrade la fonction objectif f, puis on ajoute s' à T. Quand le nombre k est atteint, chaque nouvelle solution sélectionnée remplace la plus ancienne dans la liste [3] [16] [20] .

Algorithme 1.2 - Recherche Tabou [20]

1. Générer une solution s,
2. Choisir une nouvelle solution s' qui minimise $f(s')$ dans le voisinage de s et qui ne figure pas dans la liste tabou T,
3. Si $f(s') \leq f(s)$ alors :
   $s \leftarrow s'$,
4. Poser $s = s'$ et mettre à jour T,
5. Répéter 2. 3. et 4. jusqu'à ce que le critère d'arrêt soit atteint.

Le critère d'arrêt peut être le nombre maximal d'itération sans amélioration de la solution s ou le temps limite de fonctionnement de l'algorithme qui sont fixés à l'avance.

- Méthodes évolutives (à population de solutions)

Contrairement aux méthodes partant d'une solution singulière, les métaheuristiques évolutives, débutent la recherche avec une panoplie de solutions qu'elles font évoluer au fur et à mesure des itérations. L'idée d'utiliser une population de solutions au lieu d'une seule solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité. Une grande variété de méthodes basées sur une population de solutions a été proposée dans la littérature, certaines d'entre elles ont des principes inspirés de la génétique et du comportement des insectes [29]. Ces deux phénomènes biologiques ont servi de modèle pour plusieurs algorithmes, nous citons :

Les algorithmes de colonie de fourmis

En étudiant le comportement des insectes sociaux, tel que les fourmis, tant de questions ont été posées par des chercheurs :

- Pourquoi l'essaim est-il cohérent, alors que chaque individu semble autonome ?
- Comment les activités de tous les individus sont-elles coordonnées sans supervisions ?

Les éthologistes qui étudient ce comportement observent que la coopération au sein d'une colonie de fourmis est auto-organisée. Elle semble résulter uniquement des interactions entre les individus, sans être supervisée d'une quelconque manière.
Les fourmis explorent d'abord les environs de leur nid en effectuant une marche aléatoire.

Le long de leur chemin entre la source de nourriture et le nid ou vice versa, elles déposent une substance chimique volatile appelée ***phéromone***, pour créer une piste chimique afin de marquer certains chemins favorables, guidant leurs congénères à la source de nourriture. Après un certain temps, le chemin le plus court entre le nid et la source de nourriture présente une plus forte concentration de phéromone et par conséquent, attire plus de fourmis [16].

S'inspirant de ce comportement, *Marco Dorigo* et ses collègues ont introduit l'approche dite : Optimisation par colonie de fourmis (ACO : *Ant Colony Optimization*) [21] [22] [23]. Cette métaheuristique a résolu différents problèmes d'optimisation combinatoire à forte complexité, comme le problème du voyageur de commerce, le problème de coloration de graphes ..., etc [31].

Dans les algorithmes d'optimisation par colonie de fourmis, chaque fourmi est considérée comme un agent capable de générer des solutions. La décision à prendre par une fourmi pour construire une solution, dépend de deux facteurs :

- **Le facteur de visibilité** noté $\eta_d = \frac{1}{d}$, représente l'inverse de la distance séparant le nid et la source de nourriture. Cette valeur guide le choix des fourmis vers des sources de nourriture plus proches.
- **Le facteur de trace** noté τ_d, représente la quantité de phéromone déposée sur le chemin reliant le nid à la source de nourriture. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi.

Il est également important de mettre en place un processus d'évaporation des pistes de phéromone, ce qui favorise la diversification par la prise en compte des trajets non explorés pour éviter d'être piégés dans des solutions sous optimales.

Les étapes de l'algorithme de colonie de fourmis sont décrites comme suit :

Algorithme 1.3 - Colonie de fourmis

Initialiser la trace de phéromone τ_d à 0 pour toute décision d.

Tant que (critère d'arrêt non atteint) Faire :

Pour chaque fourmi $k \in K$ Faire :

1. Construire une solution en tenant compte de la visibilité et la trace.
2. Mettre à jour la trace τ_d ainsi que la solution trouvée.

Fin pour.

Fin tant que.

Les Algorithmes Génétiques (évolutionnaires)

Les algorithmes génétiques développés en 1975 par *Holand*, ont été efficacement utilisés pour résoudre plusieurs problèmes d'optimisation en imitant le processus d'évolution naturelle des espèces dans un environnement donné, suivant le modèle Darwinien selon lequel, les individus de chaque espèce les mieux adaptés à leur environnement, ont plus de chance de survivre et de se reproduire au fil des générations par la transmission des meilleurs caractéristiques qui ont permis leurs survie, donnant ainsi des descendants de plus en plus adaptés à leurs environnement.

Les algorithmes génétiques sont des méthodes qui font évoluer un ensemble de solution appelé *population* qui représente l'espace de solution d'un problème donné, chaque point de cet espace est représenté par un *chromosome* appelé aussi *individu*. L'ensemble des variables X qui forment un chromosome (solution) sont appelés *gènes*. A chaque individu est associée une valeur dite *fitness* qui mesure son adaptation à son environnement (à l'objectif visé).

La particularité de ces algorithmes est qu'ils utilisent un codage des paramètres et non pas les paramètres eux même.

Un algorithme génétique commence par générer une population aléatoire P de N individus qui évoluent sur plusieurs générations, dans cette évolution les générations successives des différentes populations préservent une taille constante de N individus.

Le processus d'évolution d'un algorithme génétique est basé sur trois opérateurs principaux : la sélection, le croisement et la mutation [5] [7] [28].

Nous présentons ces opérateurs sous l'hypothèse que le codage des paramètres est binaire où les individus sont représentés sous forme de chaines de bits (qui peuvent prendre les valeurs 0 ou 1) comme l'illustre la figure suivante :

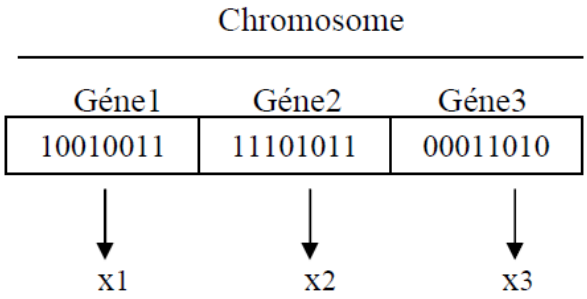

FIGURE 1.1 – Illustration du codage des variables d'optimisation

Les opérateurs génétiques :

– **Opérateur de sélection**

La sélection est un processus qui consiste à choisir parmi tous les individus de la population ceux qui vont participer à la construction d'une nouvelle génération où ce choix est basé essentiellement sur les valeurs d'adaptation de chaque individu et du critère à optimiser.

Cet opérateur intervient dans deux étapes. Premièrement à l'étape de la sélection des individus parents qui vont se reproduire pour construire de nouveaux individus enfants. Ensuite, il est appliqué à la fin de chaque itération de l'algorithme pour sélectionner les individus qui vont survivre et construire la nouvelle population [38] [56].

Il existe plusieurs techniques de sélection, les principales utilisées sont :

– *La sélection par classement* : elle consiste à classer dans un ordre croissant (ou décroissant selon l'objectif) les valeurs des fonctions d'adaptation et à retenir un nombre fixé d'individus. Seuls les individus les plus forts sont conservés.

– *La sélection proportionnelle* ou *sélection par roue de loterie* : qui associe à chaque individu un secteur de la roulette proportionnelle à la valeur de sa fonction d'adaptation. Ainsi, même les individus les plus faibles ont une chance de survivre.

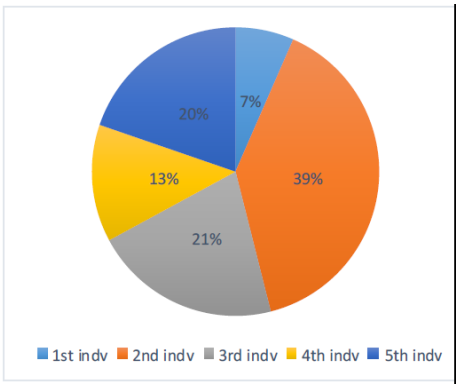

FIGURE 1.2 – Sélection par roulette

– **L'opérateur croisement**

Le croisement est appliqué avec une probabilité P_c à deux individus *parents* préalablement sélectionnés, qui échangent des parties de leurs chaînes, pour former deux nouveaux individus *enfants* [1] [38].

Usuellement, le croisement se fait selon deux modes :

– *Le croisement en un point* : consiste à choisir aléatoirement un point de coupure identique pour les chromosomes des deux parents, par la suit les sous chaînes situées après ce point sont inter-changées pour former les deux enfants.

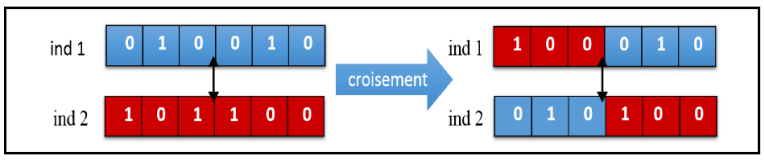

FIGURE 1.3 – Le croisement en un point

– *Le croisement en deux points* :Dans ce type de croisement, deux points de coupure sont choisis au hasard et le contenu entre ces points est inter-changé pour former les chaînes des descendants. Cet opérateur est généralement considéré comme plus efficace que le précédent.

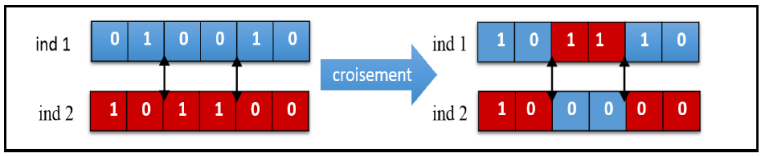

FIGURE 1.4 – Le croisement en deux points

– **L'opérateurs de mutation**

Selon une probabilité P_m, les descendants sont mutés, en modifiant aléatoirement la valeur d'un ou plusieurs composants de leur chaînes. Il ne crée pas forcement de meilleures solutions au problème, mais garantit la diversité de la population et évitent l'établissement de populations uniformes incapables d'évoluer [54].

La mutation classique consiste à remplacer dans un chromosome binaire un bit par son inverse.

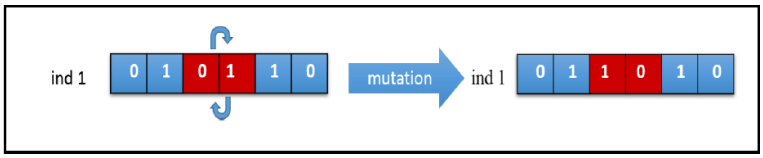

FIGURE 1.5 – une mutation binaire

Une version classique d'un algorithme génétique peut être décrite comme suit :

Algorithme 1.3 - Génétique [29]

1. Générer aléatoirement une population initiale P de N individus,
2. Évaluer chaque individu de la population P,
3. Sélectionner des individus pour la reproduction,
4. Appliquer l'opérateur de croisement sur les individus sélectionnés,
5. Appliquer l'opérateur de mutation sur les enfants obtenus,
6. Évaluer l'adaptation des enfants obtenus et des mutants,
7. Sélectionner des parents qui seront remplacés par leurs enfants,
8. Effectuer le remplacement pour obtenir une population de N individus
9. Répéter les étapes de 3. à 8. jusqu'à ce que le critère d'arrêt fixé soit atteint.

On estime que l'algorithme converge vers l'optimum (global) après un nombre maximal de générations fixé à l'avance, ou lorsque la population n'évolue plus assez rapidement.

1.4.3 Méthodes hybrides

Les approches hybrides forment une autre classe des méthodes de résolution des problèmes d'optimisation combinatoire. On parle d'hybridation dès qu'il y a combinaison d'au moins deux approches dans le but de tirer profit de leurs points forts et d'améliorer la qualité des solutions obtenus. Plusieurs types d'hybridations sont possibles, nous citons : l'hybridation de méthodes exactes-exactes, l'hybridation de méthodes heuristiques-exactes et l'hybridation de méthodes heuristiques-heuristiques, cette dernière représente la partie la plus large de toutes les hybridations [3] [26].
Ces approches hybrides fournissent de bons résultats au prix d'une complexité sans cesse accrue.

Conclusion

Dans ce chapitre, nous avons présenté d'une façon introductive, quelques notions sur l'optimisation combinatoire et ses problèmes classiques, ainsi que les métaheuristiques les plus répondues.
Vu que notre travail porte sur l'application des métaheuristiques pour la résolution des jeux, nous avons jugé nécessaire de consacrer le prochain chapitre à la présentation des éléments essentiels de la théorie des jeux.

CHAPITRE 2

NOTIONS FONDAMENTALES DE LA THÉORIE DES JEUX

Introduction

La théorie des jeux est une branche des mathématiques appliquées, qui fut développée principalement par *John Von Neumann* en 1920, et formalisée pour la première fois en 1944 par *Von Neumann* lui même et *Oscar Morgenstern* dans leur ouvrage intitulé "Theory of Game and Economic Behavior". Depuis, elle est considérée comme une nouvelle discipline appliquée dans de multiples domaines : économie, militaire, biologie, politique, ...etc.
Dans ce chapitre, nous allons présenter quelques éléments de base de cette discipline, dite **théorie des jeux**.

2.1 Notions de base et définitions

On définit plus communément la théorie des jeux comme étant une discipline théorique qui traite des objets mathématiques bien définis, elle permet de comprendre formellement des situations dans lesquelles des agents (joueurs), interagissent.

Jeu : Un jeu est une situation d'interaction stratégique entre des agents (*joueurs*) qui sont conduits à faire des choix parmi un certain nombre d'actions possibles (*stratégies*), où *l'utilité* de chaque joueur dépend non seulement de ses décisions, mais également de celles prises par les autres joueurs [2].
Les jeux peuvent donc décrire plusieurs situations réelles telle qu'une partie de poker, une vente aux enchères, la sélection des espèces, la formation d'une coalition gouvernementale ou une négociation au sein d'une organisation.
Les éléments essentiels intervenants dans la formulation d'un jeu sont :

Les joueurs : Tout agent participant au jeu et capable de prendre une décision est appelé *"joueur"*. Un joueur peut être une personne, une entreprise, un gouvernement, un consommateur, un virus..., qui vise à maximiser son utilité, en agissant selon le principe de rationalité [13].
Généralement, on note par $I = \{1, ..., N\}$, l'ensemble des joueurs qui participent au jeu, où $N \geqslant 2$.

Les stratégies : Une stratégie est la spécification complète du comportement d'un joueur dans n'importe quelle situation d'interaction. La stratégie d'un joueur représente un ensemble d'instructions à sa disposition, qui lui indiquent les actions à choisir dans toutes les situations possibles [34] [57].
Usuellement, on utilise le mot stratégie et action de manière équivalente dans le cas d'un jeu simultané [32].
On distingue deux types de stratégies :

Stratégie pure : C'est un plan d'actions qui prescrit l'action à choisir pour un joueur $i \in I$ à chaque fois qu'il est susceptible de jouer. Si ce dernier dispose de plusieurs actions m_i, alors on notera par $S^i = (s_1^i, s_2^i, ..., s_{m_i}^i)$ l'ensemble de ses stratégies pures.
Où $|S^i| = m_i$ représente le nombre de stratégies du joueur i [34].

Stratégie mixte : C'est une distribution de probabilité α^i définie sur l'ensemble des stratégies pures du joueur $i \in I$, on notera :

$$\Delta_{m_i} = \{\alpha^i = (\alpha_1^i, \alpha_2^i, ..., \alpha_{m_i}^i) \in \mathbb{R}^{m_i}, \sum_{j=1}^{m_i} \alpha_j^i = 1, \alpha_j^i \geqslant 0, \forall j = \overline{1, m_i}, i = \overline{1, N}\}$$

L'ensemble des stratégies mixtes du joueur $i \in I$, où α_j^i est la probabilité que le joueur i joue sa stratégie pure $s_j^i \in S^i$ [34].

La fonction d'utilité : Elle est associée à chaque joueur, et représente ses préférences sur son ensemble de stratégies, étant donnés les choix de ses adversaires.
Mathématiquement :

$$U^i : S = \prod_{i=1}^{N} S^i \longmapsto \mathbb{R}$$
$$s \in S \longmapsto U^i(s)$$

Avec $s = (s^1, s^2, ..., s^N)$ est une issue du jeu.
Dans ce qui suit, nous utiliserons également la notation $s = (s^i, s^{-i})$ pour désigner une issue du jeu, où s^i est la stratégie choisie par le $i^{ème}$ joueur, et s^{-i} représente la combinaison des stratégies choisies par ses adversaires [15] [57].

2.2 Représentation d'un jeu

En vertu des définitions que nous avons introduites précédemment, on peut scinder les formes de jeux en deux catégories, *Normale (Stratégique) et Extensive*, qu'on définit comme suite :

2.2.1 La forme normale d'un jeu

Cette forme est utile pour décrire des situations dans lesquelles les joueurs jouent en même temps.

En stratégies pure, elle s'écrit de la manière suivante :

$$< I, \{S^i\}_{i \in I}, \{U^i\}_{i \in I} > \tag{2.1}$$

Où ;

I : L'ensemble des joueurs.

S^i : L'ensemble des stratégies pures du joueur $i \in I$.

U^i : La fonction d'utilité du joueur $i \in I$.

De manière analogue, on peut écrire la forme normale d'un jeu en stratégies mixtes comme suite :

$$< I, \{\Delta_{m_i}\}_{i \in I}, \{E_i\}_{i \in I} > \tag{2.2}$$

Dont les éléments sont défini comme suite :

Δ_{m_i} : L'ensemble des stratégies mixtes du joueur $i \in I$.

E_i : Le gain espéré du joueur $i \in I$.

2.2.2 La forme extensive d'un jeu :

La forme extensive consiste à représenter un jeu sous forme d'un arbre, dit arbre de *Kuhn*. Il s'agit d'un modèle où les joueurs choisissent séquentiellement leurs stratégies [15]. Cet arbre est un graphe connexe et sans cycle, comme illustré dans la Figure 2.1.

Où :

- à chaque nœud terminal correspond une issue du jeu.
- un joueur est associé aux nœuds non terminaux d'un même niveau lui indiquant son tour de jouer.
- chaque arc représente chacune des actions que ce joueur peut prendre à ce nœud.

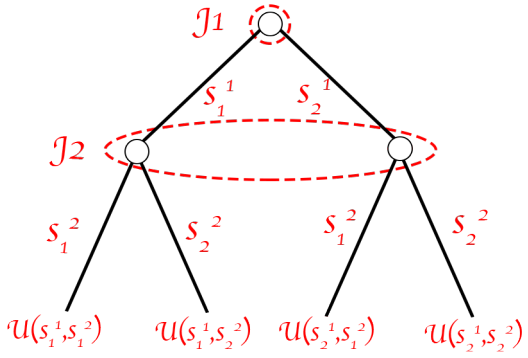

FIGURE 2.1 – Forme extensive

2.3 Classification des jeux

2.3.1 Jeu à somme nulle/non-nulle

Le jeu sous forme normale **(2.1)** est dit à **somme nulle** si :

$$\forall s \in S = \prod_{i=1}^{N} S^i, \sum_{i=1}^{N} U^i(s) = 0$$

Dans le cas contraire, le jeu est dit à **somme non nulle.** C'est à dire :

$$\exists s \in S, \sum_{i=1}^{N} U^i(s) \neq 0$$

2.3.2 Jeu à information complète/incomplète

L'information dont dispose les joueurs est dite complète, si chacun des joueurs connait la structure du jeu. C'est à dire :

- ensembles d'actions de tous les joueurs.
- gain résultant de chaque issue.

Si au moins un des joueurs ignore un des éléments de la structure du jeu, alors on parle de jeu à information incomplète.

2.3.3 Jeu à information parfaite/imparfaite

L'information dont dispose les joueurs est dite parfaite, si chacun des joueurs au moment de choisir son action, a une connaissance parfaite des actions antérieures choisies par les autres joueurs.
Si au moins un des joueurs ignore certains choix qui ont été effectués avant le sien, alors on parle de jeu à information imparfaite.

2.3.4 Jeu statique/dynamique

On dit qu'un jeu est statique, lorsque les joueurs choisissent leurs actions simultanément. C'est à dire en même temps.
On dit qu'un jeu est dynamique, lorsque les joueurs choisissent leurs actions séquentiellement. C'est à dire à tour de rôle.

2.3.5 Jeu fini/infini

On dit qu'un jeu est **fini** si, quelque soit le joueur $i \in I$, l'ensemble de ses stratégies $S^i = \{s_1^i, s_2^i, ..., s_{m_i}^i\}$ est fini. Dans le cas contraire, (s'il existe un joueur $i \in I$ tel que son ensemble de stratégies S^i est infini) le jeu est dit **infini**.

2.3.6 Jeu coopératif/non-coopératif

On qualifie un jeu de coopératif, si les joueurs (étant altruistes) peuvent former une alliance entre eux, où leurs actions seront menées conjointement de façon à atteindre un objectif commun. Dans le cas contraire (lorsque les joueurs agissent d'une façon opportuniste), le jeu est dit non-coopératif.

2.4 Concepts de solution pour les jeux non coopératifs

2.4.1 Équilibre en stratégies dominantes

Pour pouvoir définir un équilibre en stratégies dominantes, nous devons d'abord définir ces dernières [28] :

Définition 2.1. [34] Une stratégie $s^{i*} \in S^i$ est strictement dominante pour le joueur $i \in I$, ssi :

$$U^i(s^{i*}, s^{-i}) > U^i(s^i, s^{-i}), \forall s^i \in S^i, \forall s^{-i} \in S^{-i}$$

Une stratégie $s^{i*} \in S^i$ est dite faiblement dominante pour un joueur $i \in I$, ssi :

$$U^i(s^{i*}, s^{-i}) \geqslant U^i(s^i, s^{-i}), \forall s^i \in S^i, \forall s^{-i} \in S^{-i}$$
$$\text{et } \exists \overline{s^{-i}} \in S^{-i} \; et \; \overline{s^{-i}} \neq s^{-i} \; / \; U^i(s^{i*}, \overline{s^{-i}}) > U^i(s^i, \overline{s^{-i}}))$$

Par rationalité, tout joueur possédant une stratégie dominante choisirait cette stratégie qui lui procurerait un gain supérieur à celui de toute autre stratégie, quelque soit le profil de stratégies des autres joueurs. Par conséquent, si tous les joueurs choisissent des stratégies dominantes, l'issue du jeu résultante sera un équilibre pour le jeu étudié [57].

Définition 2.2. Une issue $s^* \in S$ est un équilibre en stratégies dominantes, si $\forall i \in I$, chaque composante $s^{i*} \in S^i$ est une stratégie dominante pour le joueur i, c'est à dire :

$$\forall i \in I, \forall s^i \in S^i, U^i(s^{i*}, s^{-i*}) > U^i(s^i, s^{-i*})$$

2.4.2 Recherche des équilibres par élimination itérative des stratégies dominées

Étant donné un jeu qui n'admet pas d'équilibre en stratégies dominantes, le jeu peut souvent être simplifié par un processus itératif où les stratégies dominées ne seront jamais choisies par les joueurs rationnels.

Définition 2.3. [47] Une stratégie $s^{i*} \in S^i$ est strictement (resp. faiblement) dominée pour le joueur $i \in I$, s'il existe une stratégie $\tilde{s^i} \in S^i$, telle que :

$$U^i(s^{i*}, s^{-i}) < U^i(\tilde{s^i}, s^{-i}), \forall s^{-i} \in S^{-i}$$

Respectivement,

$$U^i(s^{i*}, s^{-i}) \leqslant U^i(\tilde{s^i}, s^{-i}), \forall s^{-i} \in S^{-i}$$

Processus d'élimination des stratégies dominées

Pour pouvoir définir ce processus d'élimination, on pose les hypothèses suivantes :

- Tous les joueurs sont rationnels.
- La rationalité et le jeu sont une connaissance commune entre les joueurs.
- Un joueur rationnel ne joue jamais une stratégie strictement dominée.

En vertu de ces hypothèses, le principe du processus d'élimination consiste à éliminer les stratégies dominées pour chaque joueur. Par la suite on se retrouve avec un nouveau jeu où ses ensembles de stratégies sont réduits par rapport à ceux du jeu principal, et de nouvelles relations de dominance apparaissent. En éliminant de la même manière les éventuelles stratégies dominées pour le nouveau jeu, on se retrouve dans un autre jeu encore plus réduit. On continu le processus d'élimination des stratégies dominées jusqu'à ce que les joueurs n'aient plus de stratégies à éliminer. Si les ensembles des stratégies sont finis, et que le processus s'arrête en ne laissant pour chaque joueur qu'une seule stratégie, on dira alors que le jeu est résolvable par l'élimination itérative des stratégies dominées [47]. La procédure itérée est la suivante :

- Étape 1 : $S_0^i = S^i \forall i \in I$.
- Étape 2 : $S_1^i = \{$ stratégie non dominée $i\}$ $\forall i \in I$.
  $\vdots$
- Étape k+1 : $S_{k+1}^i = \{s^i \in S_k^i, \nexists \tilde{s^i} \in S_k^i, \forall s^{-i} \in S_k^{-i}, U^i(\tilde{s^i}, s^{-i}) > U^i(s^i, s^{-i})\} \forall i \in I$
  Stratégies non dominées face aux stratégies de S_k^i
- Étape n : $S_n^i = \bigcap_{k=1}^{n} S_k^i$, n est fini pour un jeu fini.

2.4.3 Équilibre de Nash

Les concepts que nous avons introduits précédemment peuvent ne pas donner un équilibre. En revanche, il existe une notion qui peut nous donner un équilibre dans le cas où les relations de dominance n'ont pas lieu. Ce concept a été introduit par John F.Nash dans les années cinquante.
L'équilibre de Nash représente une situation du jeu dans laquelle aucun joueur n'a intérêt à dévier unilatéralement de sa stratégie tant que les autres joueurs maintiennent les leurs. Autrement dit, une déviation unilatérale d'un joueur entrainerait une diminution du gain de celui-ci.

Équilibre de Nash en stratégies pures :

Définition 2.4. [47] Une issue $s^* = (s^{i*})_{i\in I} \in S = \prod_{i=1}^{N} S^i$ est un équilibre de Nash en stratégies pures du jeu sous forme normale **(2.1)**, si :

$$U^i(s^{i*}, s^{-i*}) \geqslant U^i(s^i, s^{-i*}), \forall s^i \in S^i, \forall i \in I \tag{2.3}$$

Équilibre de Nash en stratégies mixtes :

Définition 2.5. [47] Une issue $\alpha^* = (\alpha^{i*}) \in \Delta = \prod_{i=1}^{N} \Delta_{m_i}$ est un équilibre de Nash en stratégies mixtes du jeu sous forme normale **(2.2)**, si :

$$E^i(\alpha^{i*}, \alpha^{-i*}) \geqslant E^i(\alpha^i, \alpha^{-i*}), \forall \alpha^i \in \Delta_{mi}, \forall i \in I \tag{2.4}$$

Théorème 2.1. *[11] (Théorème de Nash, 1950) Tout jeu fini **(2.1)** admet un équilibre de Nash en stratégies mixtes.*

2.4.3.1 Stratégies de meilleure réponse :

On peut reformuler la définition de l'équilibre de Nash en introduisant la notion de meilleure réponse :
Une stratégie est une ***meilleure réponse*** d'un joueur $i \in I$ à un profil de stratégies des autres joueurs, si elle lui offre la plus grande satisfaction face à ce profil. Formellement, une stratégie de meilleure réponse est définit de la manière suivante :

Définition 2.6. [39] Une stratégie pure $s^i \in S^i$ est une meilleure réponse aux stratégies pures des autres joueurs $s^{-i} \in S^{-i}$ dans le jeu sous forme normale **(2.1)**, si :

$$U^i(s^i, s^{-i}) \geqslant U^i(\tilde{s^i}, s^{-i}), \forall \tilde{s^i} \in S^i$$

On note $MR^i(s^{-i})$, l'ensemble de toutes les stratégies pures qui sont meilleures réponses à s^{-i}. Formellement :

$$MR^i(s^{-i}) = \left\{ s^i \in S^i / U^i(s^i, s^{-i}) \geqslant U^i(\tilde{s^i}, s^{-i}) \right\}, \forall \tilde{s^i} \in S^i$$

Proposition 2.2. *[58] Une issue $s^* =\in S$ est un équilibre de Nash en stratégies pures du jeu sous forme normale **(2.1)**, ssi*

$$s^{i*} \in MR^i(s^{-i}), \forall i \in I$$

Définition 2.7. [6] Une stratégie mixte α^i est une meilleure réponse aux stratégies mixtes des autres joueurs α^{-i}, si :

$$E^i(\alpha^i, \alpha^{-i}) \geqslant E^i(\tilde{\alpha^i}, \alpha^{-i}), \forall \tilde{\alpha^i} \in \Delta_{m_i}$$

On note $MR^i(\alpha^{-i})$, l'ensemble de toutes les stratégies mixtes qui sont une meilleure réponse à α^{-i}, tel que :

$$MR^i(\alpha^{-i}) = \left\{ \alpha^i \in \Delta_{m_i} \; tq \; U^i(\alpha^i, \alpha^{-i}) \geqslant U^i(\tilde{\alpha^i}, \alpha^{-i}) \right\}, \forall \tilde{\alpha^i} \in \Delta_{m_i}$$

Proposition 2.3. *[43] Une issue $s^* = (\alpha^{i*})_{i \in I} \in \Delta$ est un équilibre de Nash en stratégies mixtes du jeu sous forme normale **(2.2)**, ssi*

$$\alpha^{i*} \in MR^i(\alpha^{-i}), \forall i \in I$$

2.4.4 Complexité du calcul de l'équilibre de nash

En 1951, Nash a prouvé que chaque jeu fini a un équilibre de Nash en stratégies mixtes, depuis, plusieurs questions ont été posées :

- Cet équilibre peut-il être atteint en pratique ?
- Existe-t-il un algorithme efficace pour trouver cet équilibre pour lequel l'existence est garantie ?... etc.

De nombreux algorithmes ont été proposé pour trouver l'équilibre de Nash, mais la question de la complexité de ces derniers n'a été résolue qu'en 1944, lorsque *Christos Papadimitriou* a introduit la classe des problèmes **PPAD** *(Polynomial Parity Arguments on Directed graphs)*, qui est une sous-classe de la classe **NP**, elle contient les problèmes dont la réponse est toujours **oui**, car une solution à ces problèmes existe toujours, même s'il pourrait être difficile de la trouver. Par la suite *Christos* a annoncé un théorème où il a prouvé que le calcul d'un équilibre de Nash pour un jeu à N joueurs, appartient à la classe de problèmes **PPAD** [19].

Conclusion

Dans ce chapitre, nous avons introduit quelques notions de base de la théorie des jeux, telles que la définition d'un jeu, d'un joueur, d'une stratégie. Ainsi que quelques concepts de solutions des jeux non coopératifs. En particulier, l'*équilibre de Nash*, qui joue un rôle fondamental dans la résolution des jeux. Le complexité de calcul de cet équilibre est la principale motivation pour de nombreux spécialistes du domaine pour engager la réflexion sur l'usage des métaheuristiques pour le calcul de l'équilibre de Nash.
Nous synthétisons dans le prochain chapitre, quelques travaux résolvant des jeux par des métaheuristiques.

CHAPITRE 3

QUELQUES EXEMPLES DE JEUX RÉSOLUS PAR DES MÉTAHEURISTIQUES

Introduction

Bien qu'il existe dans la littérature une variété de concepts de solution des jeux non coopératifs, l'équilibre de Nash reste le concept le plus important et le plus utilisé.
La complexité du calcul de cet équilibre a donné naissance au calcul des équilibres de Nash approximatifs comme solutions aux problèmes qui ne peuvent être résolus par les méthodes exactes, et ceci en explorant la piste des métaheuristiques.
Les travaux effectués au cours des trois dernières décennies, ont prouvé le succès de l'application des métaheuristiques comme approches de résolution en théories des jeux.
Nous synthétisons dans ce chapitre quelques uns de ces travaux.

3.1 Approche métaheuristique pour le calcul de l'équilibre Nash (approximatif) en stratégies pures

Le problème du calcul de l'équilibre de Nash en stratégies pures (**ENP**) a été abordé dans [24], en mettant en œuvre trois métaheuristiques : le Random Restart Hill Climbing(**RRHC**), le recuit simulé(**RS**) et le Chained Local Optimization(**CLO**) qui est une hybridation des deux algorithmes **RRHC** et **RS**. Ces métaheuristiques ont été appliquées pour la résolution des jeux non-coopératifs sous forme normale pour différentes instances du problème, même lorsque le nombre de joueurs et/ou le nombre de leurs stratégies est grand.

Dans ce travail, un nouveau concept de solution appelé *Equilibre de Nash Approximatif en Stratégies Pures* a été utilisé. Il est défini comme suit :
Soit le jeu (3.1) :

$$< I, \{S^i\}_{i\in I}, \{U^i\}_{i\in I} > \tag{3.1}$$

Où : I : est lensemble des joeurs ;
S^i : est lensemble des stratégies du joueur $i \in I$;
U^i : la fonction d'utilité du joueur $i \in I$ qu'il cherche à maximiser.
$S = \prod_{i=1}^{n} S^i$: est l'ensemble des stratégies de tous les joueurs.

Définition 3.1. On dit qu'une issue $s = (s^1, ..., s^n) \in S$ est un équilibre de Nash approximatif pour le jeu (3.1) avec une précision $\epsilon \geq 0$ si et seulement si :

$$U^i(s^i, s^{-i}) \geqslant U^i(s^{i'}, s^{-i}) - \epsilon, \forall s^{i'} \in S^i, \forall i \in I \tag{3.2}$$

Un tel équilibre est dit "$\epsilon-$équilibre de Nash" en stratégies pures.
De la définition précédente on dit qu'une issue $s^* \in S$ est un ENP pour le jeu (3.1) si et seulement si, s^* est un $0-$équilibre de Nash en stratégies pures.

3.1.1 Résolution du jeu

Pour la résolution du jeu (3.1), les auteurs dans [24] ont procedé à sa modélisation sous forme d'un problème d'optimisation combinatoire, il s'agit de minimiser la fonction suivante :

$f : S \to \mathbb{R}$

$s \mapsto f(s) = \max\limits_{i \in I} \max\limits_{s^{i'} \in S^i} \left(U^i(s^{i'}, s^{-i}) - U^i(s^i, s^{-i}) \right)$

Cette fonction f représente l'augmentation maximale de gain qu'un joueur pourrait atteindre en modifiant unilatéralement sa stratégie.

Ainsi, une issue $s^* \in S$ est un ENP si et seulement si $f(s^*) = 0$, c'est-à-dire :

$$U^i(s^{i'}, s^{-i*}) - U^i(s^{i*}, s^{-i*}) = 0 \quad \forall i \in I, \quad \forall s^{i'} \in S^i \tag{3.3}$$

Sinon, $f(s) \leq \epsilon, \epsilon > 0$ et dans ce cas, on dira que l'issue s^* est un ϵ-équilibre de Nash en stratégies pures pour le jeu (3.1).

3.1.2 Calcul de ϵ-équilibre de Nash en stratégies pures

Calculer l'équilibre du jeu (3.1) revient à chercher le minimum de la fonction objectif f, en utilisant l'un des algorithmes proposés dans [24]. La description de ces algorithmes est la suivante :

(1) L'algorithme "Random Restart Hill Climbing"

Cet algorithme commence la recherche par une solution s générée aléatoirement, et sauvegarde la meilleure solution trouvée dans son voisinage. L'algorithme recommence la recherche par une autre solution différente, si la nouvelle recherche produit une meilleure solution que celle sauvegardé à l'étape précédante, alors la solution de l'étape précédante sera remplacée par cette dernière.

Algorithme 3.1 - Random Restart Hill Climbing

Soit T le temps maximal d'exécution de l'algorithme,

1. Générer aléatoirement une issue $s \in S$,
2. $s_{best} \leftarrow s_v \in \eta_k(s)$ [1] / $f(s_v) = \max_{\tilde{s} \in \eta_k(s)} f(\tilde{s})$,
3. Générer aléatoirement une autre issue $s \in S$,
   $s \leftarrow s_v \in \eta_k(s)$ / $f(s_v) = \max_{\tilde{s} \in \eta_k(s)} f(\tilde{s})$,
4. - Si $f(s) = 0$ alors :
   s est un équilibre de Nash, retourner s.
   - Sinon aller à 5.
5. Si $f(s) < f(s_{best})$ Alors :
   $s_{best} \leftarrow s$,
6. Répéter 3. 4. et 5. jusqu'à atteindre T,
7. Retourner s_{best}.

1. Le voisinage $\eta_k(s)$ d'une issue s est l'ensembles des issues dans lesquelles au plus k joueurs changent leurs stratégies.
$k = 1$ pour le RRHC, et $k = 3$ pour le RS et CLO

(2) Le "Recuit Simulé"

Il se base sur le principe de l'algorithme 1.1 présenté dans la section 1.4.2., où la température t, est un paramètre qui influence sur la probabilité avec laquelle des solutions qui dégrade temporairement la valeur de f sont acceptées.

Algorithme 3.2 - Recuit Simulé

Soit T le temps maximal d'exécution de l'algorithme, t_0 la température initiale, et t_{leght} le nombre maximal d'itérations avant de diminuer la température,

1. Générer aléatoirement une issue $s \in S$,
2. $s_{best} \leftarrow s$ et $t \leftarrow t_0$,
3. Générer une issue $s' \in \eta_k(s)$,
4. - Si $f(s) = 0$ alors :
   s est un équilibre de Nash, retourner s.
   - Sinon, aller à 5,
5. Calculer $\Delta = f(s') - f(s)$,
6. - Si $\Delta \leq 0$ alors :
   (a) $s \leftarrow s'$,
   (b) si $f(s') < f(s_{best})$ alors : $s_{best} \leftarrow s'$,
   - Sinon, avec une probabilité égale à $e^{-\Delta/t}$ $s \leftarrow s'$,
7. Répéter 3. 4. 5. et 6. jusqu'à ce que le nombre d'itérations soit égal à t_{leght},
8. Diminuer la température selon un facteur α fixé [2],
9. Répéter 3. 4. 5. 6. et 7. jusqu'à atteindre T,
10. retourner s_{best}.

2. Lorsque la température atteint une limite inférieure à une certaine valeur ϵ, elle est réinitialisée à la température initiale.

(3) L'algorithme "Chained Local Optimization"

L'ybridation des algorithmes **RRHC** et **RS**, a donné naissance à l'algorithme **CLO**

Algorithme 3.3 - Chained Local Optimization

Soit T le temps maximal d'exécution de l'algorithme,

1. Générer aléatoirement une issue $s \in S$
2. $s \leftarrow s_v \in \eta_k(s) \ / \ f(s_v) = \max\limits_{\tilde{s} \in \eta_k(s)} f(\tilde{s})$
3. $s_{best} \leftarrow s$ et $t \leftarrow t_0$
4. Générer une issue $s' \in \eta_k(s)$
   $s' \leftarrow s_v \in \eta_k(s') \ / \ f(s_v) = \max\limits_{\tilde{s} \in \eta_k(s)} f(\tilde{s})$
5. Aplliquer les étapes de 4. à 9. de l'algorithme **RS**,
6. retourner s_{best}.

Les algorithmes présentés ci-dessus ont été exécutés sur un ensemble d'instances générées aléatoirement et comparé avec la méthode **MILP** [3]. Pour les instances de problèmes qui peuvent être résolus par le MILP, le temps requis par les métaheuristiques pour trouver un ENP est plus petit par rapport au MILP avec un écart très grand. Dans les cas où ENP n'existe pas, les petites valeurs des fonctions objectifs retrouvées par les métaheuristiques suggèrent de bons ϵ-équilibre de Nash.
Pour les jeux avec un grand nombre de joueurs et/ou d'actions, qui ne peuvent pas être résolus par le MILP, les métaheuristiques trouve toujours un ϵ-équilibre de Nash.

3. MILP est une méthode éxacte pour le calcul de ENP basé sur la programmation linéaire mixte en nombres entiers [53].

3.2 Un algorithme Tabou pour la recherche de stratégies de meilleure réponse

Dans [49], les auteurs ont proposé une approche pour la recherche de l'équilibre de Nash pour un jeu à n-joueur sous forme normale, basé sur la notion de stratégies de meilleure réponse et la recherche tabou. Cette dernière est utilisée pour éviter le problème du cyclage répétitif dans la recherche de l'équilibre de Nash.
soit le jeu :

$$< I, \{S^i\}_{i \in I}, \{U^i\}_{i \in I} > \tag{3.4}$$

Rappelons qu'une issue $s^* = (s^{i*})_{i \in I} \in S$ est un équilibre de Nash en stratégies pures pour le jeu (3.5) si et seulement si :

$$U^i(s^{i*}, s^{-i*}) \geqslant U^i(s^i, s^{-i*}), \forall s^i \in S^i, \forall i \in I \tag{3.5}$$

L'algorithme tabou de recherche des stratégies de meilleure réponse maintient une liste tabou T de longueur k qui enregistre les k dernières solutions visitées, il utilise une fonction de voisinage qui retourne toutes les issues qui peuvent être atteintes par un joueur i en changeant unilatéralement sa stratégie. À chaque itération de l'algorithme, un joueur i dévie vers une de ses stratégies de meilleures réponses.

3.2.1 Résolution du jeu

La recherche de l'équilibre de Nash dans le jeu (3.5) correspond à la recherche d'une issue $s^* = (s^{i*})$ telle que $\forall i \in I, s^{i*} \in MR^i(s^{-i})$ où : $MR^i(s^{-i})$ est l'ensemble de toutes les stratégies pures d'un joueur $i \in I$ qui sont une meilleure réponse à s^{-i}.
Les étapes de l'algorithme tabou de recherche des stratégies de meilleure réponse sont les suivantes :

Algorithme 3.4 - Algorithme tabou de recherche des stratégies de meilleures réponses
Soit le jeu (3.5) : $< I, S^i, U^i >$

1. $T \leftarrow \emptyset$,
2. Choisir aléatoirement une issue $s = (s^1, ..., s^n) \in S$,
3. $i \leftarrow$ sélection aléatoire d'un joueur,
4. $s \leftarrow ((s^{\tilde{i}}), s^{-i})$ / $s^{\tilde{i}} \in MR^i(S^{-i})$,
5. Si $|T| = k$ alors :
   supprimer l'élément le plus ancien dans la liste T,
6. $T \leftarrow T \cup \{s\}$
7. Répéter 3. 4. 5. et 6. jusqu'à atteindre une issue $s^* \in S$ qui satisfait la relation(3.3)

3.3 Un algorithme génétique pour la résolution d'un jeu de clustering

3.3.1 Construction du jeu

Avant d'entamer la modélisation et l'adaptation de l'algorithme génétique pour la résolution d'un jeu de clustering de données, nous auront besoin des définitions suivantes :

Définition 3.2. [33] Le clustering (ou partitionnement), est une méthode d'analyse de données dont le but principal est d'organiser un ensemble d'objets en groupes en fonction de leur similitude. C'est à dire que les objets similaires seront regroupés dans un même cluster (groupe), tandis que les objets dissimilaires seront chacun dans un cluster différent. Étant donné un ensemble $D = \{O_1, ..., O_i, ..., O_n\}$ de n objets, telque $O_i = (o_{i1}, o_{i2}, ..., o_{i\delta})^T \in \mathbb{R}^\delta$. Le partitionnement de D en un nombre K de groupes homogènes appelés *Clusters* est un ensemble $\xi = \{C_1, ..., C_K\}(K \leqslant n)$ vérifiant les relations **(3.7)** suivantes :

$$\begin{cases} C_i \neq \emptyset, \forall i \in \{1, ..., K\} \\ C_i \bigcap C_j = \emptyset, \forall i, j \in \{1, ..., K\}, i \neq j \\ \bigcup_{i=1}^{K} C_i = D. \end{cases} \tag{3.6}$$

Dans [33], le problème de clustering à été modélisé sous forme d'un jeu de formation de coalitions du type Δ [4]construit sous forme normale, noté par :

$$J =< I, \{S^i\}_{i \in N}, \{U^i\}_{i \in N} > \tag{3.7}$$

Où :

- $I = \{O_1, ..., O_n\}$ est l'ensemble des joueurs (objets).
- $S^i = \{C \subseteq N | O_i \in C\}$ représente l'ensemble des stratégies qui sont définies comme étant les coalitions $\{C_i \in C, i \in I\}$ auxquelles les joueurs peuvent appartenir.

Chaque issue du jeu $s = (s^1, ..., s^n)$ engendre une partition, $P^s = \{C \subseteq N | O_i, O_j \in C \leftrightarrow s_i = s_j\}$ des objets de I en k clusters. Les objets faisant le même choix vont former un cluster.

- $U_i : S^n \rightarrow \mathbb{R}^2$ est la fonction vectorielle des gains des joueurs.

avec

$$U^i(s^i, s^{-i}) = (U^{i,1}(s^i, s^{-i}), U^{i,2}(s^i, s^{-i})) \tag{3.8}$$

où :

$U^{i,1}(s^i, s^{-i}) = Connc(C_i)$: représente le premier objectif à maximiser. Il consiste à évaluer le degré dans lequel les objets voisins sont placés dans le même cluster. il est défini par la relation suivante :

$$Connc(C_i) = \frac{1}{|C_i|} \sum_{h=1}^{|C_i|} \frac{\sum_{j=1}^{L} \chi_{O_h^i, O_{vh_j}}}{L}$$

4. Un je de formation de coalitions de type Δ est défini par : un ensemble de joueur I ; un ensemble des stratégies de chaque joueur : $S^i = \{s \subset I : i \in S\}$; une fonction gain qui associe à chaque situation du jeu un gain ; pour chaque profils de stratégie $s = \{s^1, ..., s^n\}$ la coalition qui va se former est :$\wp^s = \{x \subseteq I : i, j \in x \Leftrightarrow s_i = s_j\}$

Où :

$\chi_{O_r,O_s} = \begin{cases} 1, \text{ si } O_r, O_s \in C_i \\ 0, \text{ sinon.} \end{cases}$ O_{vh_j} : est le j^{me} plus proche voisin de $O_h^i \in C_i$ et L leurs nombre.

$U^{i,2}(s^i, s^{-i}) = \frac{1}{I_A(C_i)}$: est le second objectif à minimiser. Son but est d'éviter une perte importante en terme d'inertie intra-cluster [5] , avec le critère :

$$I_A(C_i) = \frac{1}{|C_i|} \sum_{w \in C_i} d(w, ch_i)$$

Où :

$d(w, ch_i)$: est la distance entre l'objet w et le représentant ch_i.

Pour réduire le nombre de clusters initiaux, et donc la complexité du problème et pouvoir atteindre rapidement le clustering optimal, les auteurs ont posé les hypothèses suivantes :

- Si deux objets voisins ont un même ensemble de stratégies, ils seront regroupés pour en faire qu'un seul objet (joueur).
- Étant donné qu'un problème de formation de coalitions est modélisé sous forme d'un jeu non-coopératif monocritère, alors en agrégeant les deux fonctions objectifs, $Connc(C_{\widehat{i}})$ et $I_A(C_{\widehat{i}})$, la résolution analytique se fera en une seule fonction objectif $\widehat{U}$.

De ce fait, le nombre de joueurs $\widehat{n}$ sera inférieur au nombre d'objets dans I, le nouvel ensemble des joueurs devient $\widehat{I} = \{O_{\widehat{1}}, O_{\widehat{2}}, ..., O_{\widehat{n}}\}$, tel que $\widehat{n} < n$, et l'ensemble des stratégies des joueurs devient alors :

$$\widehat{S} = \bigcup_{\widehat{i}=1}^{\widehat{n}} \widehat{S}_{\widehat{i}}^{j}, \text{ tels que } j \in \{1, ..., n\} \text{ et } \forall O_{\widehat{i}} \subset I, \forall O_h \in O_{\widehat{i}} : S_h = S_j = \widehat{S}_{\widehat{i}}^{j}$$

La fonction objectif prendra alors la forme suivante :

$$\widehat{U}_{\widehat{i}}(x_{\widehat{i}}, x_{\_\widehat{i}}) = Connc(C_{\widehat{i}}) * \frac{1}{I_A(C_{\widehat{i}})}, \widehat{i} \in \{\widehat{1}, ..., \widehat{n}\} \tag{3.9}$$

5. Critères d'évaluation de la qualité de clustering, plus les clusters sont homogènes, plus l'inertie intra-cluster est faible.

Ainsi, nous aurons la nouvelle forme normale du jeu définie par :

$$\widehat{J} =< \widehat{I}, \{\widehat{S}_{\widehat{i}}\}_{O_{\widehat{i}} \in \widehat{I}}, \{\widehat{U}_{\widehat{i}}\}_{O_{\widehat{i}} \in \widehat{I}} > \tag{3.10}$$

3.3.2 Recherche de l'équilibre de Nash du jeu en utilisant des algorithmes génétiques

Pour la recherche de l'équilibre de Nash du jeu (3.8), les auteurs ont proposés un algorithme génétique dénommé $ClusGA$, dans lequel un nouveau schéma de codage représenté dans la figure (3.1) a été utilisé, où l'ordre des gènes d'un chromosome correspond aux joueurs, la valeur de chaque gène correspond à une stratégie du joueur courant et la solution engendrée par ce chromosome représente une partition possible des objets en K clusters, selon les choix des joueurs.

Position :	$\hat{1}$	$\hat{2}$	$\hat{3}$	...	$\hat{n}$
Génotype :	$\{\hat{1},\hat{2}\}$	$\{\hat{1},\hat{2}\}$	$\{\hat{2},\hat{3},\hat{5},\hat{n}\}$	...	$\{\hat{3},\widehat{n-1},\hat{n}\}$

FIGURE 3.1 – Codage d'un chromosome (solution) constitué de $\hat{n} < n$ gènes (joueurs)

Dans l'algorithme $ClusGA$, la valeur de la fonction fitness de chaque individu est calculée en utilisant la formule

$$\varphi(\wp) = R^2(\wp) * Connc(\wp) \tag{3.11}$$

Où : R^2 estime la proportion de l'inertie expliquée par les clusters

Les opérateurs génétiques appliqués sont adaptés au nouveau codage, tel que :

- La sélection des parents pour la reproduction s'est faite en utilisant la méthode de la roulette, et les meilleurs individus seront préservés dans une mémoire élite, pour qu'ils ne soient pas perdus après l'application des opérateurs de croisement et de mutation.
- Le mode de croisement utilisé est le croisement en deux points.
- L'opérateur de mutation est appliqué sur deux individus choisis aléatoirement avec une probabilité p_m, qui impose à ces derniers une déviation vers leurs stratégies de

meilleure réponse.

Dans la dernière étape de l'algorithme $ClusGA$, un remplacement de certains individus de la population courante est effectué pour la création de la nouvelle population. En effet, un individu sera membre de la nouvelle population si, le nombre de cluster de la partition générée par ce dernier est inférieur où égal à K^0 (une borne supérieure sur le nombre de clusters) et si cette partition générée est différente de celle engendrée par un autre individu de la population.

L'algorithme $ClusGA$ s'arrête après un nombre fixe de générations $Ngen$.

La solution du jeu de clustering

L'équilibre de Nash correspond à la solution ayant la meilleure valeur de la fonction d'évaluation φ dans l'ensemble des solutions. Si ce dernier est vide, alors la solution ayant la meilleure valeur de φ dans la dernière génération est choisie comme solution du jeu (une approximation de l'équilibre de Nash) du jeu(3.11)

Les étapes de l'algorithme $ClusGA$ sont décrites comme suit :

Algorithme 3.5 - $ClusGA$

1. Générer aléatoirement une population initiale $Pop(0)$ de $Npop$ chromosomes,
2. Calculer la valeur de la fonction fitness $\varphi(\wp)$ de la partition engendrée par chaque individu de la population courante $Pop(t)$,
3. Sélectionner des parents pour la reproduction,
4. Appliquer l'opérateur de croisement avec une probabilité $P_c = 0,80$ sur les individus sélectionnés,
5. Appliquer l'opérateur de mutation avec une probabilité $Pm = 1$, en se basant sur le concept des stratégies de meilleures réponses,
6. Incorporer les nouvelles solutions dans la population courante afin de créer la nouvelle population $Pop(t+1)$ de même taille $Npop$,
7. $t \leftarrow t+1$,
8. Répéter les étapes de 2. à 7. jusqu'à ce que le critère d'arrêt fixé soit atteint.

Les résultats obtenus par l'algorithme $ClusGA$ ont montré la puissance des AGs pour la résolution jeux non coopératifs assez complexes.

3.4 Hybridation de l'algorithme génétique et le recuit simulé pour la recherche de l'équilibre de Nash

Les auteurs dans [30] ont proposé un algorithme hybride pour la recherche de l'équilibre de Nash dans un jeu à information complète, sous forme normale. Cet algorithme dénommé **HSAGA** est une combinaison de deux métaheuristiques, l'algorithme génétique **GA** et le recuit simulé **SA**.

Le modèle proposé

Soit le jeu :

$$< I, \{S^i\}_{i\in I}, \{U^i\}_{i\in I} > \tag{3.12}$$

Le problème de calcul de l'équilibre de Nash dans le jeu J est transformé en un problème d'optimisation combinatoire, il s'agit de minimiser la fonction suivante :

$$\begin{aligned} f : S &\to \mathbb{R}^+ \\ s &\mapsto f(s) = \sum_{i=1}^{n} \left[\max_{s^{i'} \in S^i} U^i(s^1, ..., s^{i-1}, s^{i'}, s^{i+1}, ..., s^n) - U^i(s) \right] \end{aligned}$$

La fonction f est définie comme étant la somme des regrets de tous les joueurs par rapport à l'issue s, tel que le regret d'un joueur i par rapport à s est le gain maximal que peut gagner ce dernier en déviant unilatéralement de sa stratégie s^i.

Dans ce cas, on dit qu'une issue s^* est un équilibre de Nash du jeu (3.13) si, $f(s^*) = 0$.

La solution du jeu

Les étapes de l'algorithme proposé pour la recherche du minimum de la fonction f, qui correspond à l'équilibre de Nash du jeu (3.13) sont les suivantes :

Algorithme 3.6 - *HSAGA*

1. Générer aléatoirement une population initiale Pop de N individus,
2. Calculer la fitness, $f(s)$ de chaque individu de la population courante,
3. $s^* \leftarrow s \ / \ f(s) = \max_{\tilde{s} \in Pop} f(\tilde{s})$,
4. Créer une nouvelle population $NewPop$ en appliquant les opérateurs génétiques : sélection, croisement et mutation,
5. Calculer la fitness $f(s')$ des individus de $NewPop$,
6. Pour chaque solution faire :
   Si $f(s^{i'}) > f(s^i)$ alors : $Pop^i = NewPop^i$,
   Sinon :
   - Calculer $P = e^{(f(s^i)-f(s^{i'}))/T}$ / P étant une probabilité,
   - $Pop^i = NewPop^i$ selon la probabilité P,
7. $s^* \leftarrow s \ / \ f(s) = \max_{\tilde{s} \in Pop} f(\tilde{s})$,
8. Répeter 4. 6. 7. et 7. jusqu'à ce que t_{lenght} soit atteint,
9. Diminuer la température T,
10. Répeter les étapes de 4. à 9. jusqu'à ce que le critère d'arrêt soit atteint

Les auteurs de cet article ont effectué une étude comparative de l'approche hybride proposée avec le recuit simulé et l'algorithme génétique. Les résultats de la simulation des ces algorithmes ont montré l'efficacité de l'algorithme génétique par rapport aux deux autres algorithmes en terme de temps de calcul et la qualité de solution qu'il offre. On conclut que l'hybridation des algorithmes SA et GA n'aboutit pas à une amélioration de la qualité des solutions du problème posé.

Conclusion

Dans ce chapitre, nous avons soulevé la synthèse des travaux sur la recherche de l'équilibre de Nash en utilisant les métaheuristiques. Ceci étant l'objet principal de ce mémoire, que nous corroborons dans le chapitre suivant.

CHAPITRE 4

UN ALGORITHME GÉNÉTIQUE POUR LA RÉSOLUTION D'UN JEU DE CLUSTERING

Ce n'est pas ce que vous ne savez pas qui vous cause des problèmes, mais c'est ce que vous savez avec certitude et qui n'est pas vrai. **- Mark Twain -**

Introduction

En étudiant la complexité des algorithmes de calcul des équilibres de Nash, *Julien Laumonier* a donné les résultats suivants : le calcul de l'équilibre de Nash en stratégies pures est de complexité exponentielle en nombre de joueurs, le calcul de l'équilibre de Nash des jeux à somme nulle se fait avec les méthodes de la programmation linéaire, donc il est de complexité polynomiale, le calcul de l'équilibre de Nash des jeux à somme non nulle est un problème combinatoire de la classe NP-difficile [36].

Le recours aux méthodes de résolution des problèmes d'optimisation combinatoire présentées dans le premier chapitre, s'avère le choix le plus adéquat pour le calcul de ces équilibres en un temps raisonnable.

Dans ce chapitre, nous proposons une nouvelle adaptation des algorithmes génétiques permettant le calcul de l'équilibre de Nash en stratégies pures pour un jeu non coopératif à n joueurs, représentant un problème de clustering de données. L'adaptation proposée repose principalement sur la définition de l'équilibre de Nash en tant qu'issue du jeu où

chaque joueur choisi une stratégie de meilleure réponse aux choix de ses adversaires. La description de cet algorithme génétique et une évaluation de ses performances sont données après avoir défini le problème classique de clustering et proposé un modèle de jeu correspondant.

4.1 Le problème de clustering

Le clustering est un processus qui consiste à regrouper un ensemble d'objets (physiques ou abstraits) en clusters de telle sorte que les données d'un même cluster aient des caractéristiques similaires (homogènes), et celles appartenant à des clusters distincts soient dissimilaires (hétérogènes) [37]. Formellement, ce problème est défini comme suit :

Définition 4.1. Soit $O = \{o_1, ..., o_n\}$ un ensemble de n objets. Chaque objet $o_i \in O$ est défini par un vecteur de r attributs, i.e $o_i \in \mathbb{R}^r$. Le clustering de l'ensemble O est une partition $\wp = (C_1, ..., C_k)$ de cet ensemble, vérifiant les deux relations :

$$C_j \subset O, \quad \forall j, j = \overline{1,k}, \qquad \bigcup_{j=1}^{k} C_j = O \qquad \text{et} \qquad C_i \bigcap C_j = \emptyset, \quad \forall i, j \in \{1, \ldots, k\}, \quad i \neq j.$$

Les clusters formés sont représentés par des centroïdes. Soit $c = \{c_1, ..., c_k\}$ l'ensemble des centroïdes correspondant à l'ensemble des clusters $C = \{C_1, ..., C_k\}$, i.e c_j est le centroïde du cluster C_j, $j \in \{1, \ldots, k\}$.

4.2 Le clustering en tant que jeu non coopératif

Bien qu'il existe un grand nombre de méthodes pour la résolution des problème de clustering, la difficulté d'évaluation des solutions fournies rend difficile le choix du traitement à appliquer à chaque cas spécifique. Ces dernières années, on recense un certain nombre de travaux qui abordent le problème de clustering d'un point de vue théorie des jeux. Dans cette section, nous proposons un jeu non coopératif pour la représentation du problème de clustering de données à attributs numériques.

Les éléments du modèle de jeu que nous proposons sont définis comme suit :

Les joueurs

À chaque objet $o_i \in O$ est associé un joueur $i \in I$ où $I = \{1, ..., n\}$ est l'ensemble des joueurs.

Les stratégies des joueurs

Chacun des clusters formés représente pour chaque joueur $i \in I$, une stratégie $s_i \in S^i$ où $S^i = \{C_1, ..., C_k\}$.

Une issue du jeu $s = (s^1, ..., s^n) \in S = \prod_{i=1}^{n} S^i$, représente une partition $\wp^s$ des n objets de l'ensemble O en k clusters.

Les utilités des joueurs

La fonction d'utilité de chaque joueur $i \in I$ est définie par la relation :

$$U_i(s^i, s^{-i}) = \begin{cases} d(o_i, c_{s^i})^2\big(1 + \frac{1}{|C_{s^i}|-1}\big), & \text{si } (o_i \in C_{s^i} \text{ et } |C_{s^i}| > 1), \\ d(o_i, c_{s^i})^2\big(1 + \frac{1}{|C_{s^i}|}\big), & \text{sinon.} \end{cases} \tag{4.1}$$

où :

- c_{s^i} est le centroide du cluster (stratégie) choisi par le joueur i
- d est la fonction de similarité entre un objet o_i et le centroîde c_{s^i} du cluster qu'il a choisi. Elle est exprimée en terme de distance euclidienne et définie par la relation :

$$d(o_i, c_{s^i})^2 = \sum_{h=1}^{r} (c_{s^i}^h - o_i^h) \tag{4.2}$$

Avec

$$c_{s^i}^h = \frac{1}{|c_{s^i}|} \sum_{o_l \in c_{s^i}} o_l^h, \quad \forall h = \overline{1, r}, \quad \forall s^i \in S^i \tag{4.3}$$

Le gain d'un joueur i est inversement proportionnel à cette distance, c'est à dire que plus cette distance est grande, plus le gain du joueur i est petit, donc chaque joueur i cherche à minimiser sa fonction d'utilité définie par la formule ci-dessus.

Remarque 4.1. *Le terme* $(1 + \frac{1}{|c_{s^i}|-1})$ *figurant dans la fonction d'utilité est un paramètre qui a été choisis pour favoriser les clusters de grande taille et assurer la convergence des dynamiques de recherche des stratégies de meilleures réponses vers un équilibres de Nash.*

Ainsi, la forme normale du jeu de clustering proposé est donnée par :

$$G =< I, \{S^i\}_{i\in I}, \{U^i\}_{i\in I} > \tag{4.4}$$

Le concept de solution approprié

Le jeu (4.4) est un jeu non coopératif à n-joueurs et à somme non nulle. Le concept de solution utilisé pour sa résolution est l'équilibre de Nash.

Rappelons qu'en utilisant la notion de stratégies de meilleure réponse, une issue $s^* = (s^{i*})_{i\in I} \in S$ est dite " équilibre de Nash en stratégies pures" pour le jeu (4.4), si et seulement si :

$$\forall i \in I, s^{i*} \in MR^i(s^{-i}) \tag{4.5}$$

où :

$$MR^i(s^{-i}) = \{s^i \in S^i \ / \ U^i(s^i, s^{-i}) \geqslant U^i(\tilde{s^i}, s^{-i})\}, \forall \tilde{s^i} \in S^i \tag{4.6}$$

4.3 Calcul de l'équilibre de Nash pour le jeu G

Pour calculer l'équilibre de Nash en stratégies pures pour le jeu de clustering de données précédemment défini, nous avons proposé une adaptation d'un algorithme génétique, basée sur un processus itératif de recherche des stratégies (pures) de meilleures réponses. Dans ce qui suit, nous décrivons les principales étapes de notre algorithme nommé *BR-GA* (pour Best-Response Genetic Algorithm).

4.3.1 Codage des chromosomes

Pour la représentation des chromosomes, nous avons opté pour le codage par "numéro du groupe", car c'est le plus convenable pour représenter une solution au problème de clustering. Ainsi, chaque gène représente un objet (joueur), la valeur du gène indique le cluster auquel l'objet appartient(i.e la stratégie choisie par le joueur). Dans ce codage, un individu est finalement représenté par un seul chromosome, qui correspond à une partition possible des n objets en k clusters [44] [48].

4.3.2 Initialisation de la population initiale

Cette étape de l'algorithme BR-GA permet de générer les N individus de la population initiale. Pour générer un individu, k centroïdes sont sélectionnés aléatoirement parmi les n objets de l'ensemble O. Les objets restant sont affectés vers les clusters dont les centroïdes sont les plus proches (toujours en termes de distance euclidienne). La répétition de cette instruction N fois permet d'obtenir les N individus de la population initiale. Le pseudo-code de cette procédure est le suivant :

Algorithm 4.1 INITIALISATION

Entrées : D : Base de données - K : Nombre de cluster - Npop : Taille de la population - Ngen : Nombre de générations.

Sorties : CentPopsol - Popsol

Pour $v = 1$ **à** $Npop$ **faire**

Générer aléatoirement K centroïdes, puis les sauvegarder dans CentPopsol[v].

Affecter tous les objets restants vers le centroïde le plus proche selon la valeur minimisant la distance euclidienne (4.2), puis enregistrer ces affectation dans Popsol[v]

Mettre à jour CentPopsol[v] en utilisant la relation (4.3) .

Fin Pour

Retourner CentPopsol, Popsol

4.3.3 Evaluation des individus

Chaque individu d'une population représente une solution au problème de clustering. Par conséquent, l'évaluation d'un individu donné revient à évaluer un partitionnement des données en clusters. En clustering, il existe plusieurs mesures et critères qui permettent d'évaluer une solution, le choix parmi ces critères dépend des informations à priori que nous avons sur les données à regrouper.

Dans l'algorithme BR-GA, nous avons choisi le critère "inertie totale" en tant que fonction fitness qui permet d'évaluer les individus. L'inertie totale mesure la concentration des objets autour du centroïde. Celle-ci doit être aussi petite que possible pour minimiser la

dispersion des objets autour des centroîdes.

Formellement, elle représente la somme des inerties intra-clusters. En considérant la distance euclidienne comme une mesure de similarité, l'inertie totale peut être calculée par l'équation :

$$f(v) = \sum_{j=1}^{k} \sum_{o_i \in c_{s^i}} d(o_i, c_{s^i})^2 \tag{4.7}$$

où v est un individu quelconque d'une population donnée.

4.3.4 La sélection

Cette phase de l'algorithme a pour but la sélection des individus qui subiront les différents opérateurs génétiques (croisement et mutation). La sélection se base sur la principe de la "roue de loterie". Contrairement à cette dernière où un individu peut être sélectionné plusieurs fois, dans l'algorithme BR-GA, il ne peut être sélectionné qu'une seule fois. Comme le critère de l'inertie totale est à minimiser, la probabilité de sélection d'un individu v_i est donnée par :

$$P(v_i) = \frac{\frac{1}{f(v_i)}}{\sum_{i=1}^{N} \frac{1}{f(v_i)}} \tag{4.8}$$

La sélection se fait de la manière suivante :

- On calcule pour chaque individu v_i sa probabilité cumulée $q(v_i) = \sum_{j=1}^{i} P(v_j)$ et on tire aléatoirement un nombre $x \in [0, 1]$,
- L'individu v_i est sélectionné si $q(v_i) > x$, sinon un autre individu v_j tel que $(i + 1 \leq j \leq N)$ et $q(v_i) < x < q(v_j)$ sera sélectionné.

Cette procédure est donnée par le pseudo-code suivant :

Algorithm 4.2 SELECTION

Entrées : CentPopsol - Popsol

Sorties : Parents

Pour $v = 1$ **à** $Npop$ **faire**

Calculer $P(v_i)$ avec la relation (4.8).

Fin Pour

Tant que le nombre d'individus à sélectionner n'est pas atteint **faire**

Calculer $q(v_i) = \sum_{j=1}^{i} P(v_j), \qquad \forall i, i = \overline{1, N}$

Générer aléatoirement $x \in [0, 1[$

Si $q(v_i) < x$ **alors**

Sélectionner l'individu v_i.

Sinon

t← i

Tant que $t < N$ **faire**

$z \leftarrow q(v_{t+1})$

Si $z > x$ **alors**

Sélectionner l'individu v_{t+1}

Sinon

$t \leftarrow t + 1$

Fin Si

Fin Tant que

Fin Si

Fin Tant que

Retourner Liste des individus sélectionnés (Parents)

4.3.5 Le croisement

Une fois la sélection terminée, les individus sont répartis en couples sur lesquels on applique l'opérateur de croisement pour créer de nouveaux individus enfants. Dans le travail effectué dans ce mémoire, la probabilité de croisement P_c est fixée à $0,8$.
La méthode utilisée est le croisement en deux points générés aléatoirement, tel que :

- avec une probabilité $p \in [0, \alpha_1]$ on échange les sous-chaînes situées à gauche des points de coupures,
- avec une probabilité $p \in]\alpha_1, \alpha_2[$ on échange les sous chaînes situées entre les deux points de coupures,
- avec un une probabilité $p \in [\alpha_2, 1]$ on échange les sous chaînes situées à droite des deux points de coupures.

Les paramètres $\alpha_1 \in]0, 1[$ et $\alpha_2 \in]0, 1[$ vont être fixés expérimentalement.

Cette procédure est décrite par le pseudo-code suivant :

Algorithm 4.3 CROISEMENT

Entrées : Parents

Sorties : Enfants

Fixer la probabilité de croisement P_c et les probabilité α_1, α_2

Tant que il y'a une paire d'individus sélectionnés **faire**

Générer aléatoirement $P \in [0; 1]$

Si $Pc \leqslant P_v$ **alors**

Générer aléatoirement deux points de coupure sur la longueur d'un individu.

Générer aléatoirement $c \in [0; 1]$

Si $(c \leqslant \alpha_1)$ **alors**

Effectuer le croisement entre les deux parents du couple sélectionné en échangeant les deux séquences situées à gauche du premier point de coupure de chaque parent.

Sinon Si $c \in]\alpha_1, \alpha_2]$ **alors**

Effectuer le croisement entre les deux parents du couple sélectionné en échangeant les deux séquences situées entre les deux points de coupure de chaque parent.

Sinon

Effectuer le croisement entre les deux parents du couple sélectionné en échangeant les deux séquences situées à droite du deuxième point de coupure de chaque parent.

Fin Si

Sinon

Ne pas appliquer le croisement.

Fin Si

Fin Tant que

Mise à jour de centroîdes.

Retourner Enfants

4.3.6 La mutation

Cet opérateur génétique est très important pour l'algorithme BR-AG, car il consiste à modifier la valeur d'un ou de plusieurs gènes de l'individu muté. Une mutation consiste à imposer une modification sur deux gènes, en d'autres termes une déviation séquentielle et unilatérale de deux joueurs (choisis aléatoirement) vers une de leurs stratégies de meilleure réponse. L'opérateur de mutation est appliqué en deux étapes. La première est l'étape de mutation des enfants, la deuxième est la mutation des individus de la génération courante. À chaque étape de mutation la valeur du gène correspondant au joueur sélectionné sera modifiée par le numéro du cluster qui représente sa stratégie de meilleure réponse.

Remarque 4.2. *Avant la mutation des parents, le meilleur individu est sauvegardé.*

Cette procédure est représentée par le pseudo-code suivant :

Algorithm 4.4 Recherche des stratégies de meilleures réponses

Entrées : Enfants/Individus

Sorties : EnfantMuts/IndividuMuts

Pour $j = 1$ **à** k **faire**

$G_i^j \leftarrow$ Gain du joueur i engendré par chacune de ses stratégies.

Fin Pour

$s \leftarrow$ Stratégie actuelle du joueur i.

$min \leftarrow \arg\min_j G_i^j$.

Si $min \neq s$ **alors**

Muter le joueur i

Fin Si

Retourner EnfantMuts/IndividuMuts

Algorithm 4.5 MUTATION DES ENFANTS

Pour $v = 1$ **à** $SizEnf$ **faire**

Sélectionner aléatoirement un joueur (objet) i.

EnfantMut(v , i) ← **Algorithme 4.4.**

Tant que Le joueur sélectionné n'a pas de stratégie de meilleure réponse **faire**

Sélectionner aléatoirement un autre joueur i'.

EnfantMut(v , i') ← **Algorithme 4.4.**

Fin Tant que

Fin Pour

Mise à jour des centroïdes.

Répéter cette procedure pour la mutation d'un second joueur de l'individu v différent du premier

Retourner EnfantMuts

Algorithm 4.6 MUTATION DES INDIVIDUS DE LA SOLUTIONS INITIALE

Pour $v = 1$ **à** $Npop$ **faire**

Sélectionner aléatoirement un joueur (objet) i.

IndividuMut(v , i) = **Algorithme 4.4**

Tant que Le joueur sélectionné n'a pas de stratégie de meilleure réponse **faire**

Sélectionner aléatoirement un autre joueur i'.

IndividuMut(v , i') = **Algorithme 4.4**

Fin Tant que

Fin Pour

Mise à jour des centroïdes.

Répéter cette procedure pour la mutation d'un second joueur de l'individu v différent du premier

Retourner IndividuMuts

Après avoir terminé l'étape de la mutation, nous calculons la valeur de la fonction fitness pour chaque mutant obtenu.

4.3.7 Le remplacement

L'étape de remplacement consiste à garder dans la nouvelle génération, les N meilleurs individus parmi les enfants, les parents, les mutés et les mutants. La sélection de ces derniers se fait selon les valeurs de leur fonction fitness.

4.3.8 L'algorithme BR-AG

Avant de donner le pseudo-code de notre algorithme BR-GA, nous aurons besoin de cette procédure d'amélioration à laquelle BR-GA fait appel.

Algorithm 4.7 AMELIORATION DE LA SOLUTION_1_AG

Entrées : Newgen - Nombre d'itérations

Sorties : Solution_2_BR-GA

NbrIter ← Nombre d'itération pour la procédure d'amélioration.

$NI \leftarrow 0$

Tant que $NI \leqslant NbrIter$ **faire**

Chercher des joueurs potentiellement favorables à la mutation.

Tant que Il reste encore des joueurs i à muter **faire**

$Solution\_2\_BR - GA \leftarrow$ **Algorithme 4.4.**

Mise à jour des centroîdes

Inertie ← Calcul de l'inertie avec la relation **4.5**

Si $Inertie < Solution\_2\_BR - GA$ **alors**

$Solution\_2\_BR - GA \leftarrow Inertie$

Fin Si

$i \leftarrow i + 1$

Fin Tant que

$NI \leftarrow NI + 1$

Fin Tant que

Retourner Solution de l'algorithme BR-GA

Maintenant que chaque étape de l'algorithme a été bien détaillée, nous donnons le pseudo-code de l'algorithme principal.

Algorithm 4.8 PRINCIPAL

Entrées : D : Base de données - K : Nombre de clusters - $Npop$: Taille de la population - $Ngen$: Nombre de générations.

Sorties : Inertie optimale.

 # INITIALISATION

Individus ← **algorithme 4.1 (Initialisation)**

 # CREATION DE LA $1^{ère}$ GENERATION

Ng =0

Tant que Ng < Ngen **faire**

$Best$ ← le meilleur individu de la génération actuelle.

Parents ← **Algorithme 4.2 (Sélection).**

Coupleparents ← **Algorithme 4.3 (Croisement).**

EnfantMuts ← **Algorithme 4.5 (Mutation des Enfants).**

IndividuMuts ← **Algorithme 4.6 (Mutation des Individus).**

EnfantVidu ← [EnfantMut ; IndividuMut]

Inrtie_Optimale ← Evaluation(fitness) des EnfantVidu avec la relation **(4.5).**

EnfantVidu ← EnfantVidu + $Best$

Inrtie_Optimale ← Inrtie_Optimale $+f(Best)$

 # Construction de la nouvelle génération

Pour $v = 1$ **à** $Npop$ **faire**

Newgen ← les meilleurs individus parmi les EnfantVidus de sorte que deux individus ayant la même fitness, un seul sera pris

Inertie_Optimale ← Inertie_optimale des EnfantVidus.

Fin Pour

Individus/PopSol ← Newgen.

$Ng \leftarrow Ng + 1$

Fin Tant que

$Solution\_1\_BR - GA \leftarrow \min(Inertie\_Optimale)$.

$Solution\_2\_BR - GA \leftarrow Solution\_1\_BR - GA$

Amélioration ← **Algorithme 4.7**

Retourner Inertie_Optimale

4.4 Résultats expérimentaux

Afin d'évaluer les performances de l'algorithme BR-GA, ce dernier a été implémenté sous l'environnement MATLAB sur une machine *Intel Core i5 - Bi-coeur à 1.6 GHz - 8G de RAM* sous Mac OS 10.12. Une série d'expériences ont été menées sur un ensemble de données réelles et synthétiques. Pour juger la qualité des résultats obtenus, nous avons comparé ces derniers aux résultats fournis par Kmeans [1], l'un des algorithmes les plus connus pour la résolution des jeux de clustering. Les résultats de ce dernier sont obtenus par un simple appel de la fonction Kmeans prédéfinie sous matlab.

4.4.1 Description des bases de données utilisées

Nous avons choisi d'examiner l'efficacité de notre algorithme sur huit bases de données. Une brève description de ces bases de données en fonction du nombre d'objets n, du nombre d'attributs r et du nombre de clusters k pour chaque base est donnée dans le tableau suivant :

[2]

Types	Bases de données	n	r	k
Synthétiques	Data_3_2[2]	76	2	3
	Dim064	1024	64	16
	Dim256	1024	256	16
	Spherical_5_2[2]	250	2	5
	Spherical_6_2[2]	300	2	6
Réelles	Iris	150	4	3
	Thyroid	215	5	2
	Glass	214	9	7
	Ecoli	336	7	8

TABLE 4.1 – Jeux de données numériques utilisés pour l'expérimentation de BR-GA

1. La fonction Kmeans prédéfinie sous MATLAB utilise par défaut la puissance carrée des distances entre objets et centroîdes, comme mesure de similarité

2. http ://www.isical.ac.in/%7Esanghami/data.html

4.4.2 Réglage des paramètres

L'une des difficultés d'utilisation d'un algorithme génétique réside dans le choix de nombreux paramètres qui le contrôlent. Il s'agit du nombre d'itérations pour l'algorithme, de la taille de la population d'une part et des probabilités de croisement et de mutation d'une autre part. Le réglage de ces paramètres est crucial et conditionne la qualité des résultats obtenus. Cependant, un réglage optimal reste une tâche difficile.
Après plusieurs exécutions de l'algorithme avec différentes valeurs des paramètres, nous avons fixé la probabilité de croisement à $Pc = 0.8$ et dans le but d'imposer à un des joueurs pouvant améliorer son gain, une déviation vers une de ses stratégies de meilleure réponse, la probabilité de mutation est fixée à $Pm = 1$. Quant aux deux autres paramètres (*Npop & Ngen*) ils peuvent être modifiés à chaque exécution de l'algorithme pendant la phase d'initialisation.

4.4.3 Partie expérimentale

Pour comparer les deux algorithmes **BR-GA** et **Kmeans**, 10 exécutions de chacun ont été effectuées. Ainsi, nous indiquons pour chaque algorithme la valeur moyenne de l'inertie totale des solutions fournies.

Thyroid : Sur cette base de données, nous avons généré une population de $Npop = 40$ *Individus* et un nombre de générations $Ngen = 40$ générations, nous avons effectué 10 tests des deux algorithmes (BR-GA & Kmeans), dans 60% des dix cas, notre algorithme nous donne le meilleur résultat. En l'occurrence, une inertie minimale par rapport à **Kmeans**. Ces résultats sont exprimés sur la Figure 4.1.

Iris : Sur cette base de données, nous avons généré une population de $Npop = 40$ *Individus* et un nombre de génération $Ngen = 40$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), dans 90% des cas, notre algorithme échoue devant **Kmeans**. par contre, sur les 10% restants, il nous donne un même résultat que **Kmeans**. Ces résultats sont exprimés sur la Figure 4.1.

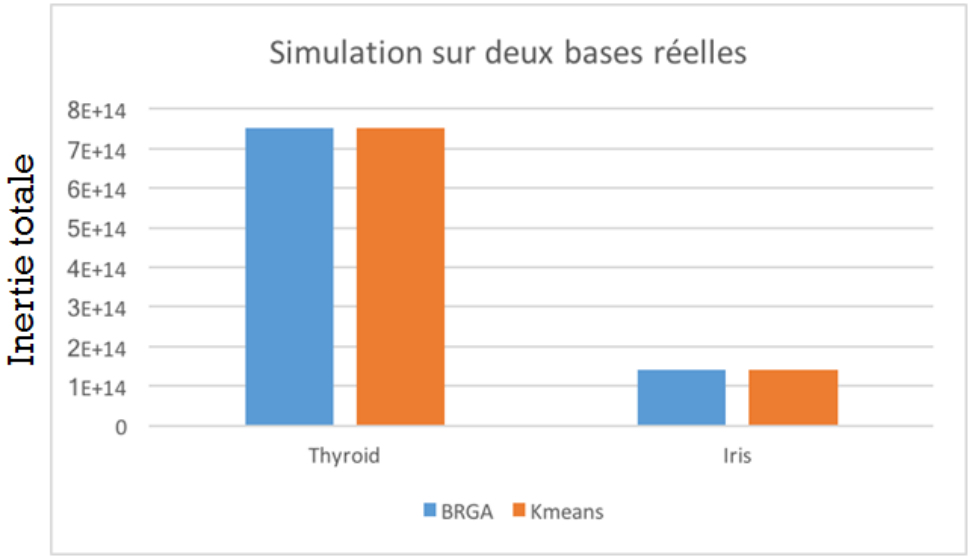

FIGURE 4.1 – Évaluation des algorithmes **BR-GA** et **Kmeans** en terme d'inertie totale sur les jeux de données réelles **Thyroid et Iris**. La petite valeur indique un meilleur résultat de clustering.

Ecoli : Sur cette base de données, nous avons généré une population de $Npop = 40$ *Individus* et un nombre de générations $Ngen = 40$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), notre algorithme échoue à 90% des dix cas devant **Kmeans**. Ces résultats sont exprimés sur la Figure 4.2.

Glass : Sur cette base de données, nous avons généré une population de $Npop = 40$ *Individus* et un nombre de générations $Ngen = 40$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), notre algorithme échoue à plusieurs reprises devant **Kmeans**. Ces résultats sont exprimés sur la Figure 4.2.

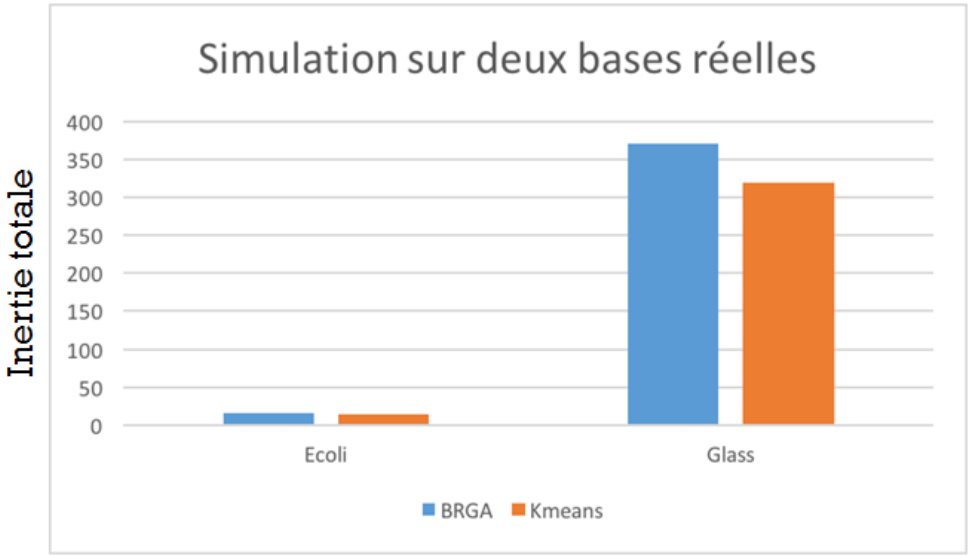

FIGURE 4.2 – Évaluation des algorithmes **BR-GA** et **Kmeans** en terme d'inertie totale sur les jeux de données réelles **Ecoli et Glass**. La petite valeur indique un meilleur résultat de clustering.

Data_3_2 : Sur cette base de données, nous avons généré une population de $Npop = 100$ *Individus* et un nombre de génération $Ngen = 100$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), dans 40% des dix cas, notre algorithme nous donne le meilleur résultat. En l'occurrence, une inertie minimale par rapport à **Kmeans**, sur les 60% restants, notre algorithme nous donne les même résultats que **Kmeans**. Ces résultats sont exprimés sur la Figure 4.3.

Spherical_5_2 : Sur cette base de données, nous avons généré une population de $Npop = 100$ *Individus* et un nombre de génération $Ngen = 100$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), dans 50% des cas, notre algorithme nous donne le meilleur résultat. Une inertie minimale par rapport à **Kmeans**. Par contre, lorsque notre algorithme échoue dans les 50% restants, la différence n'est pas fatale comparée à celle de **Kmeans**. Ces résultats sont exprimés sur la Figure 4.3.

Spherical_6_2 : Sur cette base de données, nous avons généré une population de $Npop = 100$ *Individus* et un nombre de génération $Ngen = 100$ générations, nous avons fait 10

tests des deux algorithmes (BR-GA & Kmeans), dans 80% des cas, notre algorithme nous donne le même résultat que **Kmeans**. Par contre, sur les 20% restants, à 10% notre algorithme échoue avec une légère différence, tandis que les autres 10%, **Kmeans** échoue devant notre algorithme avec une différence remarquable. Ces résultats sont exprimés sur la Figure 4.3.

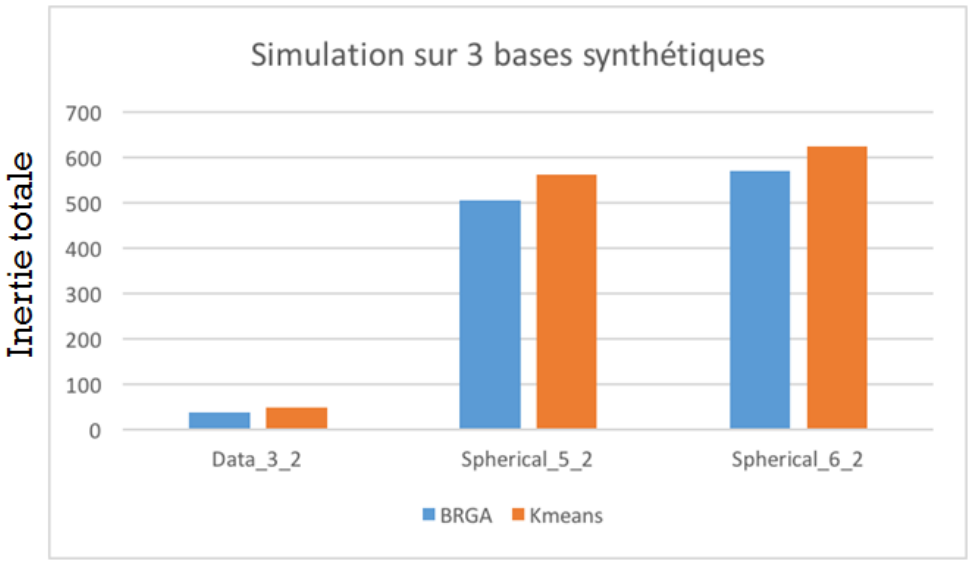

FIGURE 4.3 – Évaluation des algorithmes **BR-GA** et **Kmeans** en terme d'inertie totale sur les jeux de données synthétiques **Data_3_2, Spherical_5_2 et Spherical_6_2**. La petite valeur indique un meilleur résultat de clustering.

Dim064 : Sur cette base de données, nous avons généré une population de $Npop = 50$ *Individus* et un nombre de génération $Ngen = 50$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), presque à tous les coups, notre algorithme nous donne le meilleur résultat. Ce qui se traduit pas une inertie minimale par rapport à **Kmeans**. Ces résultats sont exprimés sur la Figure 4.4.

Dim256 : Sur cette base de données, nous avons généré une population de $Npop = 50$ *Individus* et un nombre de génération $Ngen = 50$ générations, nous avons fait 10 tests des deux algorithmes (BR-GA & Kmeans), dans 80% des cas, notre algorithme nous donne le meilleur résultat. Une inertie minimale par rapport à **Kmeans**. Ces résultats

sont exprimés sur la Figure 4.4.

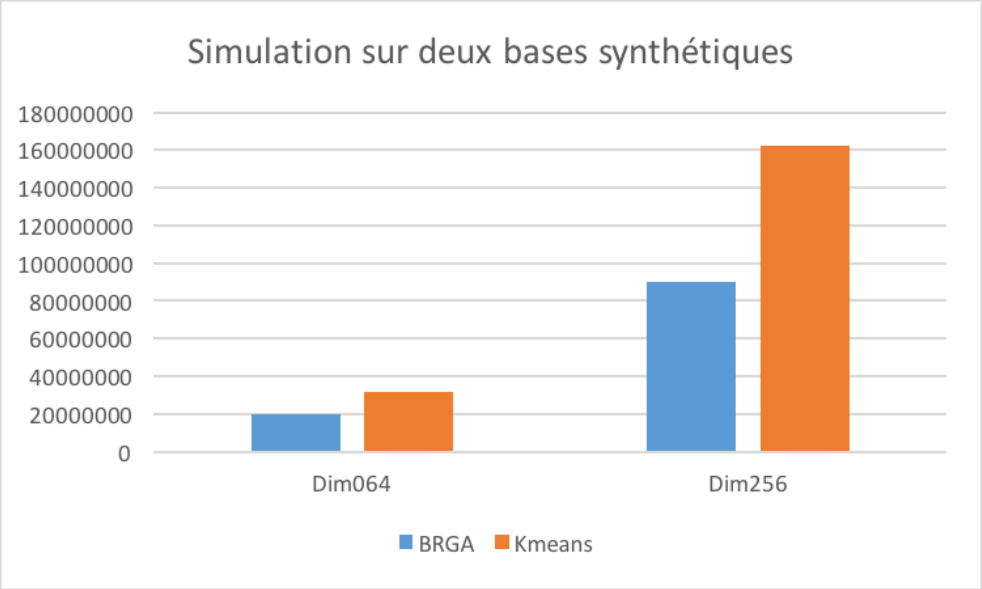

FIGURE 4.4 – Évaluation des algorithmes **BR-GA** et **Kmeans** en terme d'inertie totale sur les jeux de données synthétiques **Dim064 et Dim256**. La petite valeur indique un meilleur résultat de clustering.

Après la simulation de ces deux algorithmes sur de petites et moyennes bases de données, on a remarqué que **BR-GA** ne nous donne pas toujours l'inertie minimale comparé à l'algorithme **Kmeans** sur toutes les bases de données, mais la plupart du temps, on a une inertie égale ou légèrement supérieure à celle donnée par **Kmeans**.
Ce qui nous permet de dire que notre algorithme nous donne un bon clustering des données.

Conclusion

Dans ce chapitre nous avons abordé le problème de calcul de l'équilibre de Nash par une métaheuristique.
En effet, dans ce travail nous avons présenté un algorithme génétique pour la recherche de l'équilibre de Nash pour un jeu non coopératif à n joueurs . Nous l'avons ensuite appliqué pour la résolution d'un jeu de clustering de données. Plus précisément pour la recherche de

l'équilibre de Nash en tant qu'issue des stratégies de meilleures réponses. Une comparaison des résultats avec une méthode classique de littérature a été présentée à la fin du chapitre.

CONCLUSION GÉNÉRALE

Le travail entrepris dans le cadre de ce mémoire porte essentiellement sur le problème de calcul de l'équilibre de Nash, qui reste l'un des concepts de solution les plus utilisés dans l'application de la théorie jeux. Cependant, la recherche de cet équilibre peut s'avérer difficile.

En effet, les algorithmes dédiés au calcul de l'équilibre de Nash sont souvent inexploitables lorsqu'il s'agit de problèmes dont le nombre de joueurs et/ ou de leurs stratégies est grand.

Dans cette étude nous avons en premier lieu modélisé un problème de clustering de données sous forme d'un jeu non coopératif.

En second lieu et en se basant sur le principe des méthodes évolutionnistes, nous avons élaboré un algorithme génétique afin de résoudre le problème posé.

Une fois l'algorithme construit, nous sommes passés à son implémentation sous l'environnement MATLAB. Dans le but d'évaluer les performances de notre algorithme, nous l'avons testé sur des bases de données réelles et synthétiques.

Les résultats obtenus lors de la simulation nous permettent de certifier que notre approche donne des résultats assez satisfaisants comparés aux résultats de l'algorithme **Kmeans**.

Ce mémoire de recherche reste une opportunité pour aspirer à d'autres problématiques. En effet, ce dernier représente une base intéressante pour d'éventuelles perspectives prometteuses telles que :

- Spécifier les classes de jeux pour lesquelles **BR-GA** converge inévitablement vers un équilibre de Nash.

– Utiliser plusieurs critères et passer vers une fonction fitness.

BIBLIOGRAPHIE

[1] S.T. Abdelkader, *"Métaheuristiques pour l'optimisation des puissances actives dans un réseau d'énergie électrique"*. Mémoire de Magistère en Électrotechnique, Université de Sciences et Technologie d'Oran, 2011.

[2] M. Abdellahi, *"Modèle à deux étapes clustering-équilibrage de charges d'un réseau ad-hoc"*. Mémoire de Master, Université A.MIRA de Béjaia, 2016.

[3] M. Akli, *"Problème de tourné es de véhicules avec contraintes et fenêtre de temps"*. Thèse de magister, Département de recherche opérationnelle. Université de Mouloud Mammeri, Tizi Ouzou, 2013.

[4] M. A. Aloulou, *"Introduction aux problèmes d'ordonnancement"*. Cours. Université Paris Dauphine, 2005.

[5] K. Amari, *"Élaboration d'un algorithme génétique hybride pour l'optimisation des puissances actives dans un réseau d'énergie électrique"*. Mémoire d'ingénieur, Université Mohamed Boudiaf d'Oran, 2012.

[6] F. Barache, *"Sur la théorie des jeux évolutionnaire et ses applications en économie"*. Mémoire de Magistère, Université A.MIRA de Béjaia, 2007.

[7] N. Barnier, P. Brisset, *"Optimisation par algorithme génétique sous contraintes"*. Article. École Nationale de l'Aviation Civile.

[8] A. Belaid. *"Algorithmique Avancée"*. Cours de Master1. Département de Recherche Opérationnelle, Université A.MIRA de Béjaia, 2014.

[9] L. Belhoul, *"Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée"*. Thèse doctorat en informatique, Université Paris-Dauphine, 2014.

[10] S. Ben Ismail, *"Introduction à l'optimisation combinatoire"*. Cours. 2012.

[11] N. Berline et al, *"Introduction à la théorie des jeux répétés"*. Livre. Les éditions de l'école polytechnique, 2007.

[12] E. Bonzon, *"Modélisation des interactions entre agents rationnels : les jeux booléens"*. Thèse de Doctorat, Université Paul Sabatier de Toulouse III, 2007.

[13] M. L. Boucenna, *"Coopération dans les réseaux ad hoc par application de la théorie des jeux"*. Thèse de Doctorat en Électronique, Université Constantine1, 2014.

[14] Boughani. *"Optimisation combinatoire"*. Cours de L3. Département de Recherche Opérationnelle, Université A.MIRA de Béjaia, 2012.

[15] R. Bourlès, D.Henriet, *"Théorie des jeux"*. Livre. EAO-32-O-FIST, 2017.

[16] I. Boussaid. *"Perfectionnement de métaheuristiques pour l'optimisation continue"*. Thèse de doctorat,Université Paris-est Créteil et USTHB, 2013.

[17] O. Boussaton, *"Application de la théorie des jeux à l'optimisation du routage réseau - solutions algorithmiques"*. Thèse de doctorat, Université Henri Poincaré - Nancy I, 2010.

[18] N. Carayol, *"Jeux et Équilibre de Nash"*. Cours M1 MIMSE.

[19] C. Daskalakis, A. Fabrikant, H. C. Papadimitriou, *"The Complexity of Nash Equilibria in Succinct Games"*. Article. UC Berkeley, Computer Science Division.

[20] S. Douiri, S. Elbernoussi, H. Lakhbab. *"Cours des Méthodes de Résolution Exactes Heuristiques et Métaheuristiques"*. Université Mohammed V, Rabat.

[21] Dorigo et al. *"Positive feedback as a search strategy"*. Technical Report 91-016,Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[22] Dorigo et al. *"The ant system : Optimization by a colony of cooperating agents"*. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B, 26(1) : 29–41, 1996.

[23] Dorigo et al. *"Optimization, Learning and Natural Algorithms"*. PhD thesis, Politecnico di Milano, Italy, 1992.

[24] N. Elgers, N. Dang, P. De Causmaecker, *"A Metaheuristic Approach to computing (approximate) Pure Nash Equilibria"*. 6th International Conference on Metaheuristics and Nature Inspired computing, 2016.

[25] P. Esquirol, P. Lopez., *"L'ordonnancement"*. Economica, Paris, 1999.

[26] C. Fleurent, J. A. Ferland, *"Genetic and hybrid algorithms for graph coloring"*. Annals of Operations Research, 1996.

[27] P. Fouilhoux, *"Optimisation Combinatoire : Programmation Linéaire et Algorithmes"*. Université Pierre et Marie Curie, 2015.

[28] V. Gardeux, *"Conception d'heuristiques d'optimisation pour les problèmes de grande dimension. Application à l'analyse de données de puces à ADN"*. Thèse de Doctorat spécialité Informatique, Université de Paris-est Créteil, 2011.

[29] A. Gherboudj. *Méthodes de résolution de problèmes difficiles académiques"*.Thèse de Doctorat en Informatique, Université de Constantine 2, 2013.

[30] A. Hafezalkotob, and al, *"Using and comparing metaheuristic algorithms for optimizing bidding strategy viewpoint of profit maximization of generators"*. Article universitaire. University of Economic Sciences and Islamic Azad University, Tehran, Iran, 2014.

[31] A. Hertz, and al, *"Ants can color graphs"*. Article. Journal of the Operational Research Society, 1997.

[32] S. Hamidouche, T. Idjeraoui, *"Clustering : Approche par la théorie des jeux"*. Mémoire de Master, Université A.MIRA de Béjaia, 2013.

[33] I.Heloulou, *"Application de la théorie des jeux dans les problèmes de clustering multicritères"*. Thèse de Doctorat, Université A. Mira de Béjaia, 2017.

[34] L. Idres, *"Théorie de jeux et trafic routier"*. Mémoire de Master, Université A.MIRA de Béjaia, 2012.

[35] J. Jaffray, P. Perny, *"Théorie des Jeux"*. Cours, 2006.

[36] C. D Khamoudj, *"Classification non supervisée et théorie des jeux"*. Article. LMCS, École Supérieur d'Informatique. Alger.

[37] J. Han, M. Kamber, *"Data mining : concepts and techniques"*. Management Systems (The Morgan Kaufmann Series in Data Management Systems), ISBN 1- 55860-901-6, 2006.

[38] M. Khelaf, A. Zaidi, *"Application d'un algorithme génétique pour générer le prix d'un actif boursier"*. Mémoire de Master, Université A.MIRA de Béjaia, 2007.

[39] S. Konieczny, *"Introduction à la théorie des jeux"*. Cours. Université d'Artois - Lens.

[40] M.S. Lalami. *"Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes heuristiques et parallèles"*. Thèse de doctorat. Université Toulouse 3.

[41] A. Lemouari, *"Introduction aux Métaheuristiques"*. Support de Cours. Université de Jijel, 2014.

[42] C. Mancel *Modélisation et résolution de problème d'optimisation combinatoire issus d'applications spatiales*. Thèse de doctorat, INSA de Toulouse, 2004.

[43] M. Manceny *"Réseaux de jeux : Une extension de la théorie des jeux pour la modélisation des interactions locales. Application aux réseaux de régulation génétique"*. Thèse de doctorat. l'Université d'Évry-Val d'Essonne, 2006.

[44] C. Mbohwa, M.Mutingi, *"Grouping Genetic Algorithms : Advances and Applications"*. Livre. Springer, 2016.

[45] J. Peemöller, *"A correction to Brelaz's modification of Brown's coloring algorithm"*. Communications of the ACM, 1983.

[46] L. Péridy, *"Techniques d'Optimisation Combinatoire"*.

[47] M. S. Radjef *"La théorie des jeux non coopératifs"*. Cours Master2. Département de Recherche Opérationnelle, Université A.MIRA de Béjaia, 2014.

[48] T.P. Runarsson, *"Parallel Problem Solving from Nature - PPSN IX"*. Livre. Springer Science and Business Media, 2006.

[49] A. Sureka, P. R. Wurman, *"Using Tabu BestResponse Search to Find Pure Strategy Nash Equilibria in Normal Form Games"*. Article universitaire. Department of Computer Science, North Carolina State University.

[50] Y. Tabourier. *"Problème d'ordonnancement à contraintes purement disjonctives"*. Revue française d'automatique, d'informatique et recherche opérationnelle. Recherche opérationnelle, tome 3, n°V3, pages 51-53. 1969.

[51] B. Tsofack Nguimeya, et al. *"Algorithmes Hybrides pour la résolution des d'un problème de voyageur de commerce"*. Article universitaire. Université de Dshang, Cameroun et Université de Yaoundé I, Cameroun.

[52] M. Widmer, *"Les métaheuristiques : Des outils performants pour les problèmes industriels"*. Article universitaire. Université de Fribourg Suisse.

[53] Z. Wu et al, *"A mixed 0-1 linear programming approach to the computation of all pure-strategy nash equilibria of a finite n-person game in normal form"*. Mathematical Problems in Engineering, 2014.

[54] M. Yildizoglu, T. Vallée, *"Présentation des algorithmes génétiques et de leurs applications en économie"*. Article universitaire. Université de Nantes, LEA-CIL et Université Montesquieu Bordeaux IV, 2001.

[55] S. Zamir, R. Laraki, *"Cours de Théorie des Jeux"*.Cours. CNRS, EUREQUA Paris 1 et LEI/CREST.

[56] N. Zerari, *"Les Algorithmes Génétiques en Maintenance"*. Mémoire de Magister, Université El Hadj Lakhder Batna, 2006.

[57] S. Ziani, L. Madi, *"La théorie des jeux en files d'attente markoviennes"*. Mémoire de Master, Université A.MIRA de Béjaia, 2012.

[58] B. Ziliotto, *"Théorie des jeux"*. Cours.

Résumé

L'équilibre de Nash est l'un des concepts de solution central en théorie des jeux non coopératifs. C'est une situation dans laquelle aucun joueur n'est incité à changer unilatéralement sa stratégie. Il est bien connu que calculer un équilibre de Nash en stratégies pures est un problème complexe. Cependant, bien que pas nombreux, il existe certains travaux dans la littérature qui ont étudié les techniques de recherche locale, plus précisément, les métaheuristiques pour calculer cet équilibre. Notre objectif à travers ce mémoire est justement de présenter dans un premier temps, quelques-uns de ces travaux. Par la suite, proposer une adaptation d'un algorithme génétique basé sur un processus itératif de recherche des stratégies (pures) de meilleure réponse, convergeant vers un équilibre de Nash pour un jeu non coopératif (à n joueurs), représentant le problème de clustering de données.

Mots-clés : Jeux non coopératifs, Métaheuristiques, équilibre de Nash, Algorithmes génétiques, Fonctions de meilleure réponse.

Abstract

Nash equilibrium is one of the central solution concepts for non-cooperative game theory. It is a situation in which no player has an incentive to deviate unilaterally from his own strategy. It has been well-known that finding pure Nash equilibrium is computationally complex. However, some few works in literature have investigated local search techniques including metaheuristics for computing such equilibrium. Through this work, we present at first a few review of this literature. Secondly, we propose an adaptation of a genetic algorithm, based on an iterative process of searching for best response strategies, converging towards a pure Nash equilibrium of a n-players non-cooperative data clustering game.

Keywords : Non-cooperative games, Metaheuristics, Nash equilibrium, Genetic algorithms, Best response dynamics.