REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

MEMOIRE
Présenté à l'université de BATNA

Faculté des sciences de l'ingénieur - Département d'informatique
Pour l'obtention du diplôme Magister en informatique

OPTION
SYSTEME D'INFORMATION ET DES CONNAISSANCES

Par

SAHA ADEL

Thème

Résolution des Problèmes Multi Objectifs à Base de Colonies de Fourmi

Remerciements

Ces 16 mois de travail dans ce mémoire furent une étape importante dans ma vie, autant sur le plan professionnel que personnel. Aussi, j'aimerais commencer ce mémoire en remerciant les personnes qui m'ont épaulé et accompagné tout au long de ce parcours.

- Je remercie Dieu tout puissant clément et miséricordieux de m'avoir soigné et aidé.
- Je remercie Dr : Maamri Ramdane, mon encadreur, de m'avoir proposé ce sujet, et de l'attention qu'il a portée à mon travail. J'ai découvert grâce à lui le monde de la recherche, et plus précisément celui de l'optimisation, dans les meilleures conditions.
- J'adresse mes sincères remerciements à Dr Belatar Brahim pour l'honneur qu'il me fait en acceptant de présider mon jury.
- Je remercie Dr Kazar Okba et Dr Bilami azzeddine pour l'intérêt qu'ils ont porté à ce mémoire en acceptant d'en être les rapporteurs.
- Merci aux doctorants de l'université de Batna, ceux de la promotion 2006-2007 mais aussi tous ceux des autres promotions, avec lesquels j'ai passé de très bons moments. Je pense notamment à Guerrouf Faycel pour son fraternité plus que pour son amitié et à Benmessahel Bilal, Graini Abid, Merzougui Kamel, Belgueliel Youcef, Benfifi Khallaf, Benaouana Tayeb, Dilekh Taher, Rouaget Wahab, Naili makhlouf, Bendiaf Messaoud , Hammani Med Said, Boumediene Salah, Aouachria, Arbouche,Bachir, Chami, Mansouri, Waboudia.
- Parallèlement à mon travail de mémoire, j'ai découvert le plaisir d'enseigner. Je remercie les enseignants avec qui j'ai eu le plaisir de travailler, et tout particulièrement Mr: Boubetra Abdelhak et Mr: Akrouf Samir pour leur confiance et leurs conseils.
- Tous les enseignants qui au long de mon cursus ont su éveiller et conforter mon intérêt pour l'informatique que ce soit ceux du centre universitaire de BBA, de l'université de Sétif ou ceux de L'université de Batna. .

Dédicaces

Je dédie ce travail à mes parents, mon père Abd Elhafid et ma chère mère pour ses encouragements et ses prières tout au long de mes études, pour tous ce qu'ils avaient fait pour avoir ce résultat.

Je le dédie à mes frères, mes soeurs, ma grande mère, les petits enfants de la famille, et à ma grande famille .

Introduction générale

Présentation

Résoudre un problème d'optimisation consiste à trouver la ou les meilleures solutions vérifiant un ensemble de contraintes et d'objectifs définis par l'utilisateur. Pour déterminer si une solution est meilleure qu'une autre, il est nécessaire que le problème introduise un critère de comparaison. Ainsi, la meilleure solution, appelée aussi solution optimale, est la solution ayant obtenu la meilleure valuation au regard du critère défini.

Les problèmes d'optimisation sont des problèmes NP-difficiles et donc ne possèdent pas, à ce jour, un algorithme général permettant de les résoudre en un temps polynomial. Ce problème de l'explosion combinatoire limite l'utilisation de méthodes exactes pour la résolution à des problèmes de petites tailles. Dans le cas de problèmes de grande taille, comme cela est souvent le cas dans les applications réelles, les méthodes approchées, qui sacrifient la complétude pour gagner l'efficacité, deviennent une alternative intéressante.

Les métaheuristiques d'inspiration biologique sont à l'origine de la plupart des métaheuriqtiques présentées ces dernières années dans la littérature pour résoudre des problèmes combinatoires. Parmi ces métaheuristiques, nous pouvons citer les algorithmes génétiques, les réseaux de neurones, le recuit simulé, l'optimisation par essaim particulière, les algorithmes à estimation de distribution et l'optimisation par colonie de fourmis qui présente le cadre de notre travail.

Depuis son apparition, les algorithmes à base de colonies de fourmis requiert de plus en plus l'attention de la communauté scientifique vu le succès qu'elle a réalisé. Elle a été appliquée à plusieurs problèmes combinatoires comme le problème du voyageur de commerce, affectation quadratique, routage de véhicules, sac à dos multidimensionnel, partitionnement de graphes,...

Vu l'importance du choix de la stratégie phéromonale qui est à la fois déterminant et pas évident en fonction du problème nous avons tenté de dégager des pistes concernant ce choix , en étudiant différentes stratégies phéromonale sur un problème classique de sélection de sous ensemble qui est largement étudié dans la littérature , le problème du sac à dos multidimensionnel mono-objectif.

Nous étendons cette étude par la suite aux cas des problèmes d'optimisation multi-objectif, où le choix d'une structure phéromonale est encore plus délicat.

Les problèmes d'optimisation sont utilisés pour modéliser de nombreux problèmes appliqués : le traitement d'images, la conception de systèmes, la conception d'emplois du temps, ... La majorité de ces problèmes sont qualifiés de difficiles, car leur résolution nécessite l'utilisation d'algorithmes évolués, et il n'est en général pas possible de fournir dans tous les cas une solution optimale dans un temps raisonnable. Lorsqu'un seul critère est donné, par exemple un critère de minimisation de coût, la solution optimale est clairement définie, c'est celle qui a le coût minimal. Mais dans de nombreuses situations, un seul critère peut être insuffisant.

En effet, la plupart des applications traitées intègrent plusieurs critères simultanés souvent contradictoires appelé communément problème multi-objectif. Intégrer des critères contradictoires pose un réel problème.

L'optimisation multi-objectif consiste donc à optimiser simultanément plusieurs fonctions. La notion de solution optimale unique dans l'optimisation uni-objectif disparaît pour les problèmes d'optimisation multi-objectif au profit de la notion *d'ensemble de solutions Pareto optimales*.

L'utilisation des métaheuristiques pour résoudre des problèmes multi-objectifs a fait l'objet d'un intérêt de plus en plus croissant. A présent, les métaheuristiques constituent un des champs de recherche les plus actifs dans l'optimisation multi-objectifs. La plupart des travaux existants concernent l'optimisation bi-objectif. Le cas multi-objectif reste difficile à résoudre, non seulement à cause de la complexité de ces problèmes, mais aussi à cause du nombre élevé des solutions efficaces à un problème d'optimisation multi-objectif.

Nous proposons, dans ce mémoire, d'étudier les capacités de la métaheuristique ACO pour la résolution des problèmes d'optimisation multi-objectif. Pour ce faire, il est indispensable de choisir la stratégie phéromonale appropriée. De plus, pour ce genre de problèmes, le choix des structures phéromonales est un autre point clé dans la résolution.

Organisation du mémoire:

L'organisation de ce mémoire est la suivante : les premiers chapitres dressent un état de l'art non exhaustif des domaines de l'optimisation multiobjectifs et les méthodes de résolution utilisées. Nous présentons l'optimisation multiobjectifs tout en introduisant des concepts fondamentaux tels que la dominance, la surface de compromis. Nous décrivant aussi les problèmes d'optimisation multiobjectifs les plus connus dans le domaine de l'industrie. Ainsi les principales approches non Pareto de résolution pour ces problèmes et quelques approches Pareto, appartenant à la classe des métaheuristiques, dédiées aux problèmes d'optimisation multiobjectifs.

Dans le troisième chapitre nous présentons quelques méthodes de résolution Pareto fondées sur des métaheuristiques pour les problèmes d'optimisation multiobjectif. Ainsi, nous introduisons les méthodes de recherche locale et les approches évolutionnaires. Nous décrivons aussi deux algorithmes évolutionnaires représentatifs adoptant une approche Pareto pour résoudre les problèmes d'optimisation multiobjectifs.

Le quatrième chapitre introduit la métaheuristique d'optimisation par colonies de fourmis, là où on parle des fourmis réelles et des insectes sociaux en générale, de l'intelligence et du comportement collective des fourmis (Auto organisation, stigmergie, contrôle décentralisé, hétérarchie dense et des pistes de phéromones). Puis en passe aux fourmis artificielles et en introduit quelques algorithmes basés sur la métaheuristique d'optimisation par colonies de fourmis.

Le cinquième chapitre concerne l'étude de différentes stratégies phéromonales qui porte sur le problème du sac à dos multidimensionnel uni-objectif (MKP). Nous présentons un algorithme fourmi générique de l'état de l'art permettant de mettre en évidence l'importance du choix de la stratégie phéromonale utilisée. Cet algorithme sera instancier en trois variantes qui présentent trois différentes stratégies phéromonales possibles.

Cet algorithme sera comparé avec d'autres stratégies phéromonales pour montrer l'importance du choix de ces stratégies sur la qualité des solutions et le temps d'exécution, et pour montrer aussi l'influence des paramètres de l'algorithme fourmi sur la qualité des solutions construites ainsi que sur la similarité de ces solutions.

Nous pensons que cette étude expérimentale de différentes stratégies phéromonales sur le MKP va nous servir de base, pour le cas multi-objectif (MOKP), dans le choix de la stratégie phéromonale, et aussi dans le choix des valeurs des paramètres de l'algorithme ACO.

Le dernier chapitre regroupe les contributions les plus fondamentales de ce travail. Nous proposons une nouvelle variante d'un algorithme générique de l'état de l'art basé sur l'optimisation par colonies de fourmis pour résoudre des problèmes d'optimisation multi-objectifs. Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de structures de phéromone considérées. Cet algorithme permet de tester et de comparer, dans un même cadre, différentes approches existantes ainsi que de nouvelles approches. Ainsi nous pouvons comparer ces approches sans se soucier du schéma ACO considéré ni des détails d'implémentation. La nouvelle variante ainsi les différentes variantes de cet algorithme sont testées par la suite sur le problème du sac à dos multi-dimensionnel multi-objectif.

Chapitre 1. Optimisation Combinatoire

1.1. Introduction :

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

L'optimisation combinatoire est minimiser (ou maximiser) une fonction souvent appelée fonction coût, d'une ou plusieurs variables soumises à des contraintes. Le sujet de l'optimisation combinatoire dans un domaine discret. Il faut trouver parmi toutes les possibilités, souvent en nombre fini, la possibilité optimale. Ceci parait facile mais devient infaisable dès que la taille du problème est suffisamment grande. La taille pour laquelle la recherche d'un optimum devient infaisable est petite, très souvent plus petite que la taille des problèmes pratiques. En général, la difficulté d'un problème grandit très vite avec le nombre des variables. Il n'est pas alors faisable d'examiner toutes les possibilités.

Prendre une décision, opérationnelle ou stratégique (par exemple, planifier une tournée de livraison), c'est déterminer les options (l'ordre et les dates de visite des clients) qui répondent à plusieurs impératifs simultanés (la durée des trajets, la capacité du camion de livraison, la disponibilité des clients, les horaires de travail du chauffeur, etc.), puis choisir parmi ces options la meilleure selon un ou plusieurs critères donnés (minimiser les coûts de transport ou livrer les bons clients au plus tôt). Généralement l'ensemble des options possibles est dénombrable mais potentiellement infini, de sorte qu'il est impossible de l'énumérer entièrement en temps raisonnable, aussi performant que soit le système de calcul utilisé. L'optimisation combinatoire est la discipline de l'étude théorique et pratique de ces problèmes de décision. Elle recouvre l'ensemble des outils analytiques, logiques et algorithmiques qui permettent la résolution de ces problèmes difficiles. Émanant des mathématiques par la recherche opérationnelle et de l'informatique par la théorie de la complexité et l’intelligence

artificielle, ces outils se partagent en grandes classes technologiques -- telles que les algorithmes de graphe, la programmation linéaire en nombres entiers, les métaheuristiques, la programmation par contraintes -- définissant ainsi différents modes et objectifs de résolution. De manière transversale, les outils se combinent en grands domaines d'application telles que l'ordonnancement, le transport, l'analyse de données, la bio-informatique, la finance, etc.

1.2. Définition (problème combinatoire)

Un problème combinatoire est toute situation dont on cherche d'avoir une solution tout en respectant la présence d'un ensemble de contraintes. La solution c'est un résultat de faire combiner ces contraintes ensemble d'une manière qu'on maximise quelques uns et on minimise les autres, ces contraintes ont une caractéristique primordiale, c'est que chaque contrainte influe sur les autres soit quand on minimise sa valeur ou on la maximise, dans un autre terme on dit que les contraintes sont conflictuelles.

Par exemple, le schéma suivant présente une situation de problème combinatoire: où on veut acheter une voiture dans la mode et en même temps avec un prix raisonnable qui ne peut pas dépasser certaine limite. Si on maximise la première contrainte (une bonne voiture) on va avoir un prix maximale, dans le contraire on va aboutir à une mauvaise voiture mais avec un prix minimale dans les limites; on constate dans cet exemple que c'est difficile d'arranger ces deux contraintes dans nos besoins.

1.3. Un problème d'optimisation :

1.3.1 Définition :

Les problèmes d'optimisation ont été définis pour représenter les problèmes décrits dans le paragraphe précédent. Ces problèmes occupent actuellement une place de choix dans la communauté scientifique. Non pas qu'ils aient été un jour considérés comme secondaires mais l'évolution des techniques informatiques a permis de dynamiser les recherches dans ce domaine.

Le monde réel offre un ensemble très divers de problèmes d'optimisation :

- Problème combinatoire ou à variables continues.
- Problèmes à un ou plusieurs objectifs.
- Problèmes statistiques ou dynamiques.
- Problème dans l'incertain.

Cette liste n'est évidemment pas exhaustive, et un problème peut être à la fois multiobjectifs et dynamique. [1]

Un tel problème est caractérisé par :

- Un problème d'optimisation est défini par un espace d'état, une ou plusieurs fonction(s) objective(s) et un ensemble de contraintes.
- L'espace d'état est défini par l'ensemble de domaines de définition des variables du problème.
- Les variables du problème peuvent être de nature diverse (réelle, entière, booléenne, etc.) et sont exprimées de données qualitatives ou quantitatives.
- Une fonction objective représente le but à atteindre pour le décideur (minimisation de coût, de durée, d'erreur, ...). Elle définit un espace de solutions potentielles au problème.
- L'ensemble de contraintes définit des conditions sur l'espace d'état que les variables doivent satisfaire. Ces contraintes sont souvent des contraintes d'inégalité ou d'égalité et permettent en général de limiter l'espace de recherche.
- Une méthode d'optimisation cherche le point ou un ensemble de points de l'espace de état possible qui satisfait au mieux un ou plusieurs critères. Le résultat est appelé valeur optimale ou optimum. [1]

On peut dire qu'un problème d'optimisation se définit comme la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont des contraintes à évoluer dans une certaine partie de l'espace de recherche.

De cette définition de problème d'optimisation, il apparaît qu'un problème multiobjectifs ou multicritères peut être définit comme un problème dont on cherche l'action qui satisfait un ensemble de contraintes et optimise un vecteur de fonctions objectives. La difficulté principale d'un problème multiobjectifs est qu'il n'existe pas de définition de la solution optimale. Le décideur peut simplement exprimer le fait qu'une solution est préférable à une autre mais il n'existe pas une solution meilleure que toutes les autres.
Ce besoin d'optimisation vient de la nécessité de l'ingénieur de fournir à l'utilisateur un système qui répond au mieux au cahier des charges. Ce système devra être calibré de manière à:

- Occuper le volume nécessaire à son bon fonctionnement (coût des matières premières),
- Consommer le minimum d'énergie (coût de fonctionnement),
- Répondre à la demande de l'utilisateur (cahier des charges).

Dans ce sens, nous pouvons définir :

1.3.1.1 ***Fonction objective :***

C'est le nom donné à la fonction f (on l'appelle encore fonction de coût ou critère d'optimisation). C'est cette fonction que l'algorithme d'optimisation va devoir « optimiser » (trouver un optimum).

Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante :

$$\begin{cases} \min \ f(\vec{x}) & (\text{fonction à optimiser}) \\ \text{avec} \quad \vec{g}(\vec{x}) \leq 0 & (\text{mcontra intes d'énigalité}) \\ \text{et} \quad \vec{h}(\vec{x}) = 0 & (\text{pcontra intes d'égalité}) \end{cases}$$

On a $\vec{x} \in \Re^n$, $\vec{g}(\vec{x}) \in \Re^m$ et $\vec{h}(\vec{x}) \in \Re^p$

Ici les vecteurs $\vec{g}(\vec{x})$ et $\vec{h}(\vec{x})$ représentent respectivement *m* contraintes d'inégalité et *p* contraintes d'égalité. [2]

1.3.1.2 Variables de décision :

Elles sont regroupées dans le vecteur $\vec{x}$. C'est en faisant varier ce vecteur que l'on cherche un optimum de la fonction ***f.***

1.3.1.3 Minimum global :

Un « point » $\vec{x}^*$ est un minimum global de la fonction f si on a :

$f(\vec{x}^*) < f(\vec{x})$ Quel que soit $\vec{x}$ tel que $\vec{x}^* \neq \vec{x}$

1.3.1.4 Minimum local fort :

Un « point » $\vec{x}^*$ est un minimum local fort de la fonction f si on a :

$f(\vec{x}^*) < f(\vec{x})$ quel que soit $\vec{x} \in V(\vec{x}^*)$ et $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un « voisinage » de $\vec{x}^*$

1.3.1.5 Minimum local faible :

Un « point » est un minimum local faible de la fonction f si on a :

$f(\vec{x}^*) \leq f(\vec{x})$ quel que soit $\vec{x} \in V(\vec{x}^*)$ et $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un « voisinage »de $\vec{x}^*$.

[2].

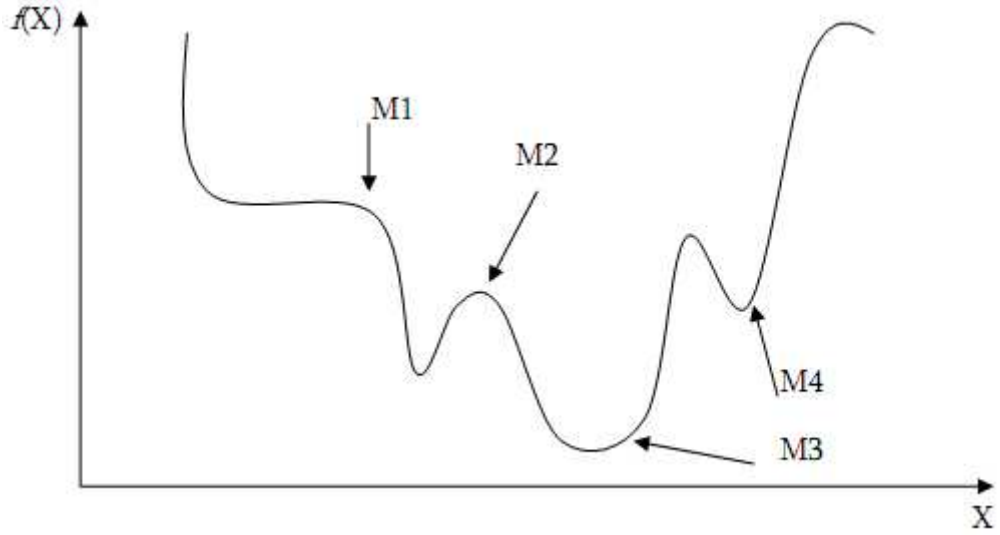


Fig 1.1. Les différents Minima [2].

En résumant ces notions mathématiques représentant un problème d'optimisation multiobjectif par la figure suivante :

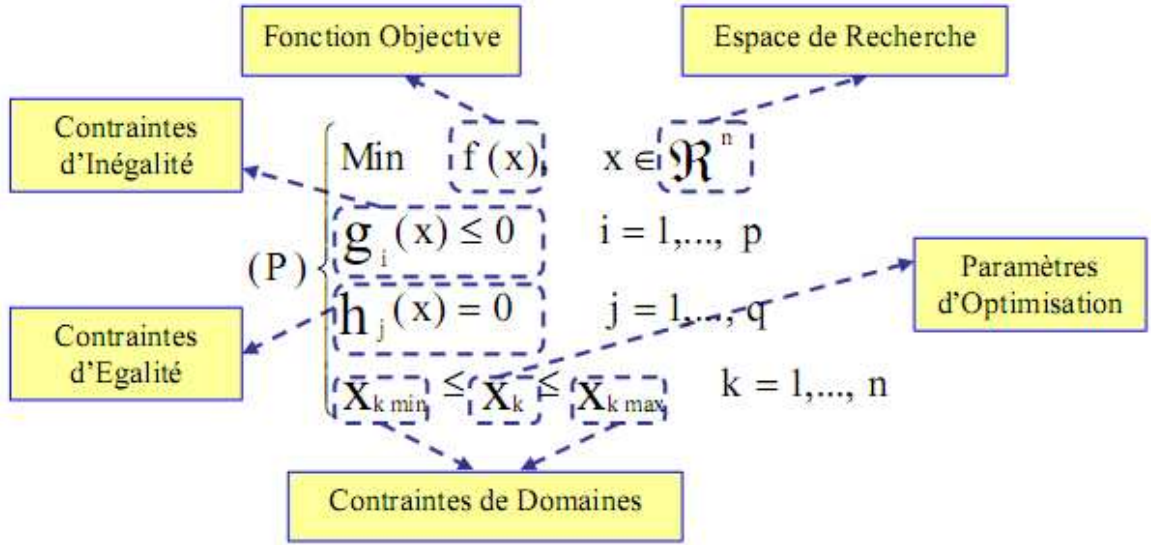


Figure 1.2.La présentation mathématique d'un problème d'optimisation multiobjectif. [3]

1.3.2 Les variables d'un problème multiobjectifs :

Les variables du problème sont les paramètres dont le concepteur doit ajustés pour modifier le système à concevoir. Il y a plusieurs types de ces variables :

1.3.2.1 Les variables indépendantes :

Sont les quantités actuelles dont le concepteur s'occupe directement, comme la géométrie, les propriétés du matériel, le volume de production, la surface finale, la configuration des composants, les propriétés de lubrification et autres …. Les variables indépendantes sont usuellement connues comme les variables de problème ou les paramètres de problème.

1.3.2.2 Les variables dépendantes :

Sont les variables dont le concepteur ne peut attribuer des valeurs mais il les utilise pendant les paramètres du problème. Les variables dépendantes sont usuellement nommées les caractéristiques ou les attributs du problème. Exemple de caractéristiques du problème l'énergie de consomption, le prix et l'erreur de contrôle. La valeur du problème est largement une fonction des caractéristiques du problème. Dans l'optimisation, la valeur de la fonction objective correspond à la valeur du caractéristique particulière. Une fonction objective c'est ainsi la relation entre les paramètres du problème et les valeurs des caractéristiques particulières. Pour un problème d'optimisation général, il faut être très difficile ou bien impossible de représenter analytiquement tous comme des relations.

Généralement, les caractéristiques doivent être le résultat de la simulation complexe, ou ils doivent inclure en quantifiable le jugement de l'homme.

1.3.2.3 Les variables d'état :

Sont le type intermédiaire des variables de problème entre les variables dépendantes et les variables indépendantes, comme la pressure de cylindre hydraulique ou le courant dans un moteur électrique. On ne peut pas attribué directement des valeurs aux variables d'état, aussi ces variables ne peuvent pas directement contribuer dans la valeur de problème comme les caractéristiques.

1.3.2.4 Les variables opérationnelles :

Sont les variables qui peuvent être changées par l'opérateur après la conception actuellement construise.

1.3.2.5 Les variables de l'environnement :

Ou les variables externes sont des facteurs écologiques qui affectent le problème au cours de l'utilisation, comme le changement de charge, la température extrême et celle d'usage. Le concepteur a pour déterminer les conditions de travail du problème les variables de l'environnement et les variables opérationnelles dans ses expériences.

Le problème peut être formulé quant à l'attribution des valeurs aux paramètres du problème pour être sûr que les variables d'état et les caractéristiques sont les meilleures au cours du champ large des variables opérationnelles et de l'environnement. Ça nécessite une optimisation de problème multiobjectif complexe. [4]

1.**3.3 La classification des problèmes multiobjectifs :**

On peut classer les différents problèmes d'optimisation que l'on rencontre dans la vie courante en fonction de leurs caractéristiques :

1- Nombre de variables de décision :
   - ✓ une → mono variable.
   - ✓ Plusieurs →multi variable.

2- Type de variable de décision :
   - ✓ Nombre réel continu →continu.
   - ✓ Nombre entier →entier ou discret.
   - ✓ Permutation sur un ensemble fini de nombres →combinatoire.

3- Type de fonction objective :
   - ✓ Fonction linéaire des variables de décision →linéaire.
   - ✓ Fonction quadratique des variables de décision →quadratique.
   - ✓ Fonction non linéaire des variables de décision →non linéaire.

4- Formulation de problème :
   - ✓ Avec contraintes → contraint.
   - ✓ Sans contraintes → non contraint.

1.4. Exemples de problèmes d'optimisation combinatoire :

1.4.1 Le voyageur de commerce

Parmi les problèmes les plus célèbres et connus dans le domaine d'optimisation multiobjectif le problème de Voyageur de Commerce. C'est un problème d'affectation assignant une route pour un véhicule entre quelques nœuds, dont l'algorithme cherche un ensemble de solutions qui répondent à quelques critères.

Le Problème du Voyageur de Commerce (Traveling Salesman Problem, TSP) possède une formulation élémentaire: étant données n villes, quel chemin doit-on emprunter pour les parcourir tout en minimisant la longueur totale du trajet? Il doit rendre régulièrement visite aux clients de son département. Sa clientèle étant répartie dans N localités différentes de sa région, y compris son domicile, il constate un jour que ses frais de route sont relativement élevés et surtout qu'ils diffèrent considérablement d'une tournée à l'autre. Il finit par s'adresser à un informaticien pour qu'il lui rédige un programme permettant de minimiser ses frais de route. Une première version de l'algorithme pourrait donc être :

(1) Minimiser les frais de route.

Cette première formulation trop vague ne permet guère de reconnaître le vrai problème à résoudre. Ainsi par exemple les données d'entrée (représentées ici par les causes des frais) ne sont pas mentionnées du tout. Une deuxième approche consiste donc à essayer de dégager des informations supplémentaires sur le champ de données en entrée.
Finalement, cette deuxième étape fait savoir à l'informaticien que la totalité des frais de route se compose :

• Des frais d'essence pour le trajet en voiture et,
• Des frais de logement pour la nuit.

Alors, il a constaté que l'action « Minimiser les frais de route » c'est « minimiser les frais d'essence et les frais de logement ». Si dans une troisième approche l'informaticien demande la raison des différences de frais, il apprend que l'ordre dans lequel les différentes localités sont visitées joue un rôle non négligeable, puis, la question est de savoir « s'il y a des clients qu'il a visités plus souvent que d'autres ? ». Cette dernière approche consiste donc à essayer de simplifier le problème initial pour lui trouver une solution acceptable, à défaut d'être optimale.
C'est le représentant de commerce qui spécifie enfin qu'il rend visite exactement une fois par mois à chaque client de sa région, qu'une tournée de visites a une durée de cinq jours, et qu'il passe les nuits dans des hôtels à même prix. Ainsi, les frais de logement pour la nuit étant fixes, l'informaticien peut en conclure que l'on peut les négliger lors de la résolution du problème de minimisation. L'informaticien finit par formuler le problème du voyageur de commerce de la manière suivante :
«Une fois par mois, à partir de son domicile, un représentant de commerce rend visite à sa clientèle qui est répartie sur N localités différentes (y compris son propre domicile) de sa région. Lors d'un même tour, il ne passe par chaque localité qu'une seule fois. C'est ainsi que son trajet se trouve divisé en exactement N parties différentes et que la totalité des frais de route s'obtient par addition des frais de route individuels, c.-à-d. des différentes parties. Chaque partie de route occasionnant des frais individuels différents, la totalité des frais varie selon l'ordre dans lequel le représentant fait sa tournée. Le problème consiste à déterminer le tour dont la somme des frais de route individuels est minimale.
En réalité, la consommation d'essence pour une partie de route n'est évidemment pas proportionnelle à la distance parcourue, mais elle dépend des conditions particulières des routes (autoroute, route nationale, route droite - route sinueuse, route plane - route raide) et notamment du sens dans lequel la route est parcourue.»

Par cette formulation de problème on déduit qu'il appartient à la classe des problèmes dits NP, ce qui signifie qu'il n'existe pas d'algorithme fournissant le chemin optimal en temps polynomial. Ce paradoxe en fait un problème stimulant et fécond qui donne lieu à des stratégies heuristiques originales et à des applications au domaine en plein essor de la bioinformatique. [5]

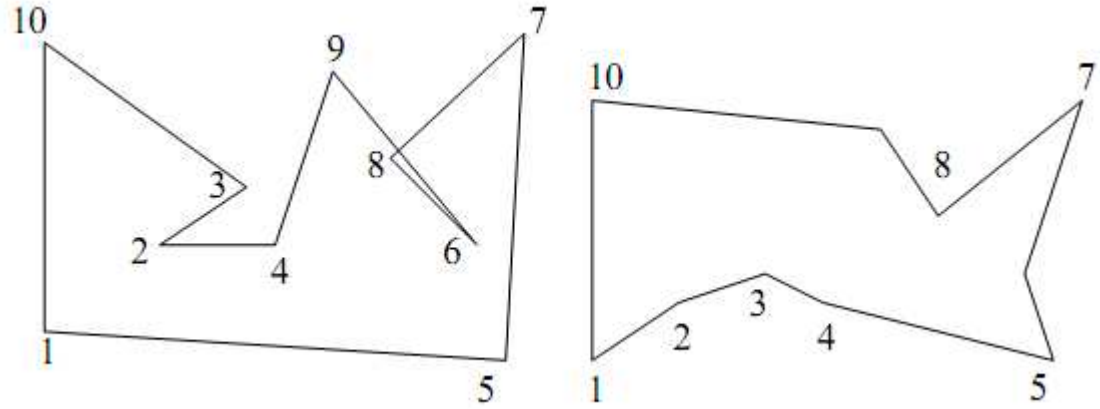


Fig.1.3.Le tour d'un voyageur de commerce dans 10 villes [6]

1.4.1.1 L'historique :

La résolution du problème PVC a passé par plusieurs étapes durant le siècle passé, en résumant ce chemin par:

19ème siècle

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19ème siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman. Hamilton en a fait un jeu : Hamilton's Icosian game : les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections prédéfinies.

Années 1930

Le PVC est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood. Une attention particulière est portée sur les connections par Menger et Whitney ainsi que sur la croissance du PVC.

L'année 1954

Solution du PVC pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du cutting-plane.

L'année 1975

Solution pour 100 villes par Camerini, Fratta and Maffioli

L'année 1987

Solution pour 532, puis 2392 villes par Padberg et Rinaldi

L'année 1998

Solution pour les 13 509 villes des Etats-Unis.

L'année 2001

Solution pour les 15 112 villes d'Allemagne par Applegate, Bixby, Chvàtal et Cook des universités de Rice et Princeton. [7]

1.4.1.2 La complexité :

Ce problème est un représentant de la classe des problèmes NP-complet.

L'existence d'un algorithme de complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en O(N!) où N est le nombre de villes.

La conception d'un algorithme correspondant qui donne une solution optimale n'est pas du tout triviale si l'on songe qu'il y a (N - 1)! = (N 1)*(N - 2)*(N – 3) … 3*2*1 tours possibles pour N localités ce qui donne pour N = 20 exactement 1 2165,1017 tournées possibles, d'autant plus qu'un tel algorithme pourrait se révéler très coûteux en temps d'exécution. [8]

En supposant que le temps pour évaluer un trajet est de 1 µs, le tableau montre l'explosion combinatoire du PCV.

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	12 µs
10	181440	0,18 ms
15	43 milliards	12 heures
20	60 E+15	1928 ans
25	310 E+21	9,8 milliards d'années

Tableau 1.1 Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes (on suppose qu'il faut 1 µs pour évaluer une possibilité) [9]

Ce qui donne plus de complexité de problème PVC c'est que le parcours des nœuds se fait de la manière que: les nœuds parcourus ne se re-parcourent pas, il faut retourner au nœud de départ et il faut minimiser le chemin parcouru à la plus petite valeur possible toute en se basant sur les distances entre les nœuds comme le montre 1.4.

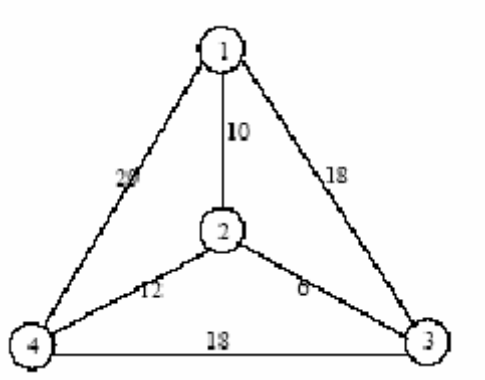


Fig. 1.4. Un graphe pour le problème de TSP [10]

1.4.1.3 Intérêt :

Le PVC fournit un exemple d'étude d'un problème NP-complet dont les méthodes de résolution peuvent s'appliquer à d'autres problèmes de mathématiques discrète.

Néanmoins, il a aussi des applications directes, notamment dans les transports et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire ou, dans l'industrie, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé (les trous représentent les villes) [11]. Ainsi que le chemin le plus cours pour le routage des paquets de messages dans un réseau informatique.

1.**4.2 Le problème du sac à dos (knapsack Problem)**

Le problème du sac à dos multidimensionnel (MKP) est un problème d'optimisation combinatoire sous contraintes NP-difficile. Ce qui explique le nombre de travaux qui ont été effectués pour résoudre ce problème [12].

Plusieurs problèmes pratiques peuvent être formalisés comme un MKP. Nous citons l'allocation des processeurs et des bases de données dans les systèmes informatiques distribués, le chargement des cargaisons, les problèmes de découpage de stock, etc.

Le MKP permet de modéliser une grande variété de problèmes où il s'agit de maximiser un profit tout en ne disposant que de ressources limitées. De manière formelle, il est présenté comme suit :

Soit

- m le nombre de ressources
- n le nombre d'objets sélectionnables
- $p_j \quad (j \in 1..n)$ le profit apporté par l'objet j
- $r_{ij} \quad (i \in 1.m, j \in 1..n)$ la consommation de la ressource i par l'objet j

– b_i $(i \in 1..m)$ la quantité totale disponible de la ressource i.

On définit le $MKP(n,m)$ par :

$$\text{MKP} \begin{cases} \text{Maximiser } \sum_{j=1}^{n} p_j x_j \\ \text{Tq } \sum_{j=1}^{n} r_{ij}.x_j \leq b_i, i \in 1..m \\ x_j \in \{0,1\}^n \end{cases}$$

x_j $(j \in 1..n)$ est une variable de décision qui peut prendre pour valeur 0 (si l'objet j n'est pas sélectionné) ou 1 (s'il est sélectionné).

1.4.3 Le problème du sac à dos multidimensionnel multi_objectifs

Ce problème est parmi les problèmes les plus étudiés dans la communauté multi-objectif. L'objectif de ce problème est de maximiser un vecteur de fonction profit tout en satisfaisant un ensemble de contraintes de capacité du sac à dos. Plus formellement, un MOKP est défini comme suit :

$$\text{Maximiser } f_k = \sum_{j=1}^{n} p_j^k x_j, \forall k \in 1..m$$

$$\text{s.c } \sum_{j=1}^{n} w_j^i x_j \leq b_i, \forall i \in 1..q$$

$$x_j \in \{0,1\}, \forall j \in 1..n$$

Où

m est le nombre de fonctions objectif,

n est le nombre d'objets,

x_j est la variable de décision associée à l'objet o_j ,

q est le nombre de contraintes de ressource,

w_j^i est la quantité de la ressource i consommé par l'objet o_j ,

b_i est la quantité totale disponible de la ressource i,

p_j^k est le profit associé à l'objet o_j relativement l'objectif k.

1.4.4 Le problème du sac à dos quadratique

Le problème du sac à dos quadratique, ou en anglais Quadratic Knapsack Problem (QKP), consiste à sélectionner un sous-ensemble d'objets dont le poids total ne dépasse pas une capacité donnée c du sac à dos et de maximiser le profit total. Plus formellement, le QKP est défini comme suit : Supposons $N = \{1..n\}$ un ensemble d'objets donné où chaque objet a un poids positif w_j . De plus, nous disposons d'une matrice d'ordre n d'entiers non négatifs $P = \{p_{ij}\}$, où le p_{ij} est un profit accompli en sélectionnant deux objets différents i et $j \in N$. En outre, le profit p_{ii} est accompli si l'objet $i \in N$ est choisi. Des variables binaires x_j sont introduites, qui indiquent si l'objet $j \in N$ est sélectionné. Ainsi, ce problème peut être formulé comme suit :

$$\begin{aligned} \text{Maximiser} \quad & \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \\ \text{tel que} \quad & \sum_{j \in N} w_j x_j \leq c \\ & x_j \in \{0, 1\}, j \in N \end{aligned}$$

1.4.5 Problème SAT (Problème de satisfaction)

Dans 1996, Yokoo [13] propose le problème SAT qui modélise les variables en deux valeurs logiques et utilise une technique d'avant vérification pour trouver l'affectation de valeur satisfaisable. Après, les auteurs divisent un algorithme de backtracking et expérimentent les heuristiques d'ordre de variables [14]. [15] furent les premiers à proposer un accélérateur reconfigurable qui implémente le backtracking avec la propagation de variables logiques [16] comme une stratégie de déduction basique. Les auteurs en plus suggérèrent deux extensions pour leur architecture avec des techniques d'analyse de conflits qui allouent pour backtracking non chronologique et l'addition de clause dynamique [17]. Platzner et De Micheli [18] présentèrent plusieurs architectures de SAT basées sur le backtracking avec 3 valeurs logiques.

Ils utilisèrent les techniques de déduction, parmi les qui ne s'intéressent pas aux variables et aux implications. Une compilation d'environnement coutume pour ces accélérateurs SAT est discutée par Mencer [19]. Abramovici et De Sousa [20] et Abramovici et Saab [21] présentèrent une architecture basée sur l'algorithme de PODEM utilisé pour la génération automatique de test de modèles. Leur architecture utilise le backtracing plutôt que le backtracking et propage le résultat requis pour le formula booléenne face au variables. [22]

Le problème de satisfiabilité propositionnelle (SAT) c'est de déterminé si une expression booléenne aurait un étiquetage satisfaisant (ensemble d'affectations vraies). Les problèmes sont usuellement exprimés dans la forme normale conjonctive : une conjonction pour les clauses $C_1\Lambda$ …. ΛC_m où chaque clause C est une disjonction de littérales l1V …Vln est chacun a une variable booléenne X ou sa négation X'. Une variable booléenne peut être étiqueté chacune vrai ou faux. Une affectation satisfaisante a au moins une littérale vraie dans chaque clause. [23]

La plupart de problèmes combinatoires ont été avec succès modélisés et résolus comme SAT, mais pas tous les problèmes ne succombent facilement aux méthodes de SAT.

Une raison c'est que les modèles de SAT peuvent s'avérer pour être très larges. Quelques chercheurs ont proposé plus de langages expressifs, partiellement, pour réduire les tailles de modèles. Des exemples incluent les équivalences nichées [24], le ou exclusive [25], les disjonctions et les conjonctions de littérales [26], les contraintes de cardinalité [27], …. En particulier, quelques modèles de SAT peuvent être réduis exponentiellement dans la taille par les reformuler aux modèles Pseudo Booléen (PB) [28]. [23]

Le meilleur problème d'expertise c'est le problème de satisfiabilité booléenne (SAT).

Donnant :

- Un ensemble de *n* variables booléenne $x_1, x_2, \ldots, x_n$,
- Un ensemble de littérales, constituées des variables xi et leurs compléments x'i, et
- Un ensemble de m propositions $C_1, C_2, \ldots, C_n$, constituées par les littérales combinées par le ou logique ou l'opérateur +.

SAT est en quête d'une affectation de valeurs vraies à des variables qui réalise la conjonction normale à partir de **CNF** $C_1 * C_2 * \ldots * C_m$ vraie, ou * dénote l'opérateur logique.

1.4.6 Problème de coloration de graphe (graph coloring)

C'est un problème parmi les problèmes NP-Complet grâce à la difficulté trouvée lors de sa résolution, il consiste d'affecter un nombre défini de couleurs k aux sommets d'un graphe non orienté d'une manière que les couleurs des nœuds adjacents sont différentes. La coloration minimale utilise le petit nombre possible de couleur (couleur chromatique). La version décisive de coloration de graphe (k-coloring) demande quels sommets dans le graphe peut être colorés en utilisant un nombre ≤ k couleurs pour un k connu. [29]

L'inapproximation du problème de coloration de graphe suggère qu'il peut être plus difficile pour le résoudre. En utilisant ce problème quelques applications sont résolues dans un temps raisonnable comme :

Les problèmes de « **Time Tabling And Scheduling** » fréquemment rejettent certaines tâches en parallèle à cause les dépendances entre computations. Planification des programmes avec le minimum hardware peuvent fréquemment être formalisée sous forme du problème de coloration de graphe.

Le problème d'allocation de registres: qui cherche à assigner des variables à un nombre limité de registres durant l'exécution de programme. Deux variables ne peuvent pas être assignées dans le même registre si se sont « vive » en même temps. L'assignement de variables en plus amène à l'exécution rapide comme moins de variables nécessitent d'être rapportées de la mémoire. Pour formaliser ce problème, on crée un graphe dont les nœuds représentent les variables et les arêtes représentent les conflits entre les variables.

Des diagrammes colorés pour l'assignement de libre conflit, et si le nombre de registres dépasse le nombre chromatique, un assignement de registre libre existe. [30]

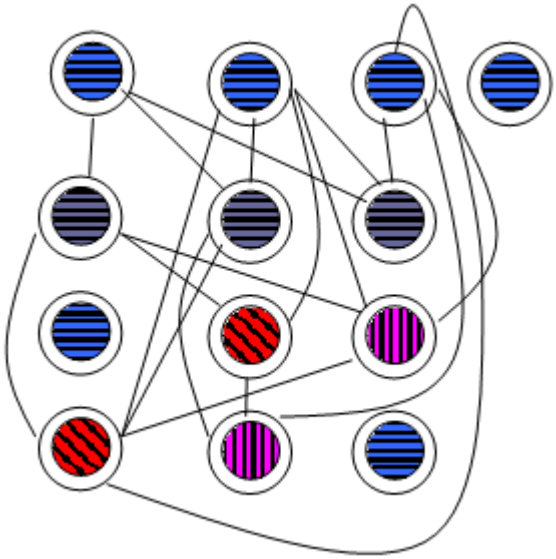


Fig 1.5. Exemple de graphe coloré

1.4.7 Problème de N-Queen

Dans le problème de 8-Queens, on a un tableau d'échec régulier (8 par 8) et huit rênes qui doivent être placées dans le tableau dans un état dont pas de deux rênes dont une touche l'autre. Ce problème peut être naturellement généralisé, qui cède facilement le problème de N-queens. Beaucoup d'approches classiques d'IA qui s'appliquent à ce problème travaillent d'une manière constructive ou incrémentale : un commence par placer une rêne et après placer les n rênes, un tente de placer la ($n^{ième\ +1}$) rêne dans la meilleure position, c'est-à-dire

une position dont la nouvelle rêne ne peut touche les autres. Typiquement, quelque sorte du mécanisme Backtracking est appliqué : s'il n'y a pas de position faisable pour la ($n^{ième+1}$) rêne, la nième est déplacée dans une autre position.

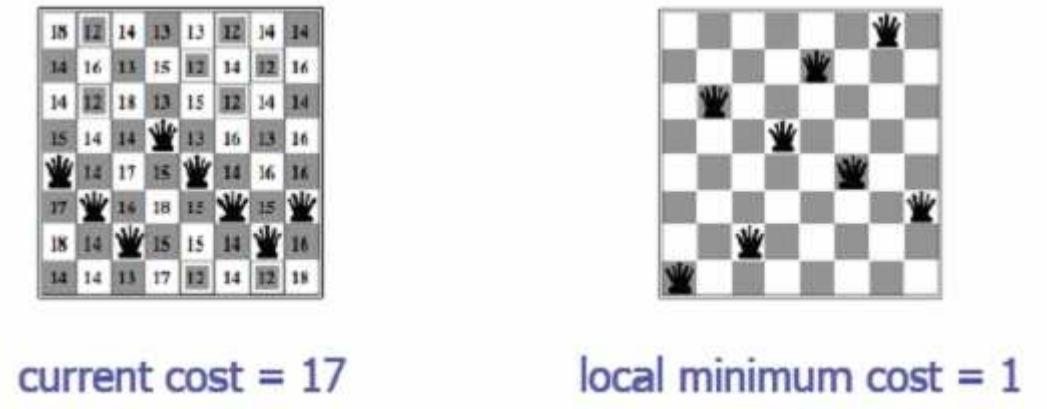


Fig 1.6. le problème de 8-Queens [31]

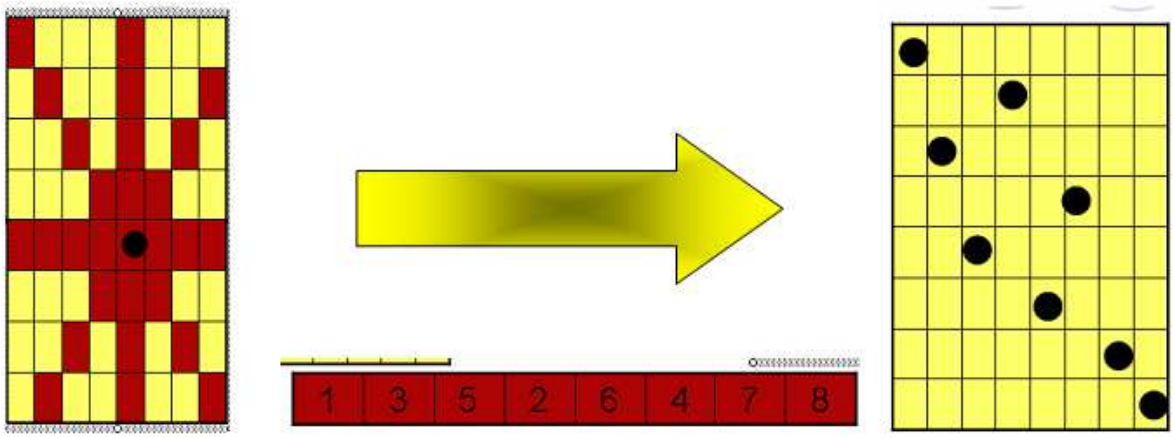


Fig 1.7. Illustration de la solution du problème de 8-Queens.

1.5. Résolution d'un problème d'optimisation combinatoire

Résoudre un problème d'optimisation combinatoire nécessite l'étude de trois points particuliers:

–la définition de l'ensemble des solutions réalisables,

–l'expression de l'objectif à optimiser,

–le choix de la méthode d'optimisation à utiliser.

Les deux premiers points relèvent de la modélisation du problème,le troisième de sa résolution.

Afin de définir l'ensemble des solutions réalisables, il est nécessaire d'exprimer l'ensemble des contraintes du problème .Ceci ne peut être fait qu'avec une bonne connaissance du

problème sous étude et de son domaine d'application. La programmation linéaire peut être utilisée à cet effet.

Le choix de l'objectif à optimiser requiert également une bonne connaissance du problème. La définition de la fonction objectif mérite toute l'attention de l'analyste car rien ne sert de développer de bonnes méthodes d'optimisation si l'objectif à optimiser n'est pas bien défini. Comme nous le verrons par la suite, il peut être très difficile de trouver un objectif unique d'optimisation .Nous pouvons donc être amené à proposer une modélisation multi-objectif.

Enfin, le choix de la méthode de résolution à mettre en œuvre dépendra souvent de la complexité du problème. En effet, suivant sa complexité, le problème pourra ou non être résolu de façon optimale. Dans le cas de problèmes classés dans la classe P, un algorithme polynomial a été mis en évidence. Il suffit donc de l'utiliser. Dans le cas de problèmes NP-difficiles, deux possibilités sont offertes.Si le problème est de petite taille,alors un algorithme exact permettant de trouver la solution optimale peut être utilisé (procédure de séparation et évaluation (Branch&Bound), programmation dynamique...). Malheureusement, ces algorithmes par nature énumératifs, souffrent de l'explosion combinatoire et ne peuvent s'appliquer à des problèmes de grandes tailles (même si en pratique la taille n'est pas le seul critère limitant). Dans ce cas, il est nécessaire de faire appel à des heuristiques permettant de trouver de bonnes solutions approchées. Parmi ces heuristiques, on trouve les métaheuristiques qui fournissent des schémas de résolution généraux permettant de les appliquer potentiellement à tous les problèmes.

Nous voyons donc ici que la phase de modélisation du problème est très importante puisque c'est elle qui permettra par exemple de reconnaître un problème de la classe P d'un problème NP-difficile. En particulier, la définition de l'objectif est cruciale mais peut être difficile à réaliser, surtout lors de l'étude de problèmes réels.

1.6. Conclusion

Nous avons défini dans ce chapitre les problèmes d'optimisation combinatoire ainsi que nous avons présenté quelques exemples.

Chapitre 02

Etat De L'art De L'optimisation Multiobjectif

2.1. Problèmes d'optimisation mono-objectifs

Résoudre un problème d'optimisation consiste à trouver une solution qui minimise ou maximise un critère particulier. Dans la plupart des cas, l'optimum découvert n'est pas unique. Ainsi il existe un ensemble de solutions minimisant ou maximisant le critère considéré. Nous pouvons décrire formellement un problème d'optimisation comme suit :
Considérons un problème de minimisation :

Soit $f : \mathbb{R}^n \rightarrow \mathbb{R}$,
et soit $\mathcal{X}$ un ensemble fermé de $\mathbb{R}^n$
alors l'ensemble des minimiseurs est : $\hat{\mathcal{X}} = \{\vec{x} \in \mathcal{X} | \forall \vec{x'} \in \mathcal{X},\ f(\vec{x'}) \geq f(\vec{x})\}$
et le minimum du problème est : $f(\hat{\mathcal{X}})$

L'ensemble X représente l'ensemble des solutions potentielles du problème. Cet ensemble est déterminé à l'aide de contraintes (souvent analytiques) données dans l'énoncé du problème. Le formalisme précédent ne faisant pas explicitement intervenir les contraintes du problème, un problème d'optimisation mono-objectif est plus souvent donné sous la forme suivante :

Minimiser	$f(\vec{x})$	(fonction à optimiser)
tel que	$\vec{g}(\vec{x}) \leq 0$	(q contraintes à satisfaire)
avec	$\vec{x} \in \mathbb{R}^n,\ \vec{g}(\vec{x}) \in \mathbb{R}^q$	

Ces deux écritures sont équivalentes, la deuxième étant le plus souvent rencontrée dans les domaines de l'Intelligence Artificielle et de l'Optimisation. Il est clair que pour minimiser (resp. maximiser) une fonction d'un problème, il faut déjà être capable de déterminer l'ensemble des solutions potentielles. On distingue ainsi deux notions : la notion d'espace de recherche représentant l'ensemble des valeurs pouvant être prises par les variables, et la notion d'espace réalisable représentant l'ensemble des valeurs des variables satisfaisant les

contraintes. Dans le cadre de ce mémoire, nous traiterons de problèmes appartenant à la classe de problèmes NP-difficiles.

Sur la figure 1.1, $\underline{x_i}$ (resp. $\overline{x_i}$) représente la borne inférieure (resp. supérieure) donnée pour x_i, C représente l'espace de recherche égal au produit cartésien des domaines des variables, et *X* l'espace réalisable délimité par les contraintes. Cette figure, illustre en dimension 2 le fait que $\mathcal{X} = \{\vec{x} \mid \vec{g}(\vec{x}) \leq 0, \vec{x} \in \mathcal{C}\}$ l'espace réalisable (délimité par des contraintes) est un sous-ensemble de C $(\text{ici } \mathcal{C} = \mathbb{R}^2)$ l'espace de recherche.
Sauf mention contraire explicite, tous les énoncés et définitions seront donnés dans le cadre de problèmes de minimisation. En effet un problème de maximisation peut etre aisément transformé en problème de minimisation en considérant l'équivalence suivante :

$$\text{maximiser } f(\vec{x}) \iff \text{minimiser } -f(\vec{x})$$

Pour la suite de ce mémoire, les vecteurs pourront être notés de manière indifférente $\vec{x}$ ou x.

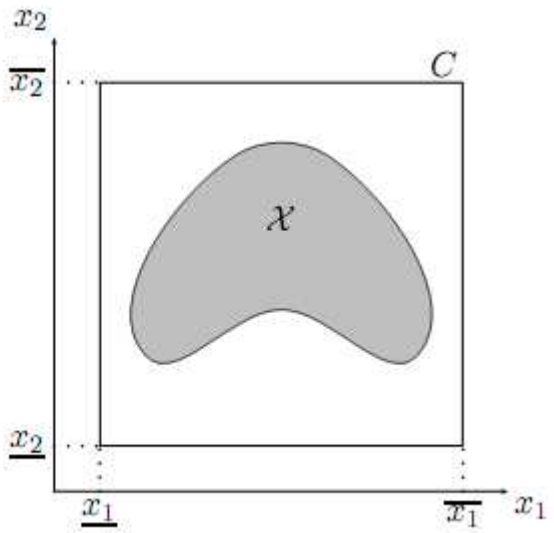


Fig. 2.1. Espace de recherche et espace réalisable

2.2 Vocabulaire et définitions

Nous donnons ici quelques définitions des principaux termes fréquemment utilisés dans ce mémoire :

Définition1: Une boite ou pavé : C'est le nom donné au produit cartésien d'intervalles fermés. L'ensemble C de la figure 2.1 est le pavé résultant du produit cartésien des intervalles [x1] et [x2].

Définition 2 : Vecteur de décision : C'est le nom donné au vecteur $\overrightarrow{x}$. Il correspond à l'ensemble des variables du problème.

Définition 3 Fonction objectif : C'est le nom donné à la fonction f (appelé aussi fonction de coût ou critère d'optimisation).

Définition 4 Minimum global : Un point $\overrightarrow{x} \in \mathcal{X}$ est un minimum global du problème si et seulement si : $\forall \overrightarrow{x'} \in \mathcal{X},\ f(\overrightarrow{x}) \leq f(\overrightarrow{x'})$ (il appartient alors à l'ensemble des minimiseurs).

Définition 5 Minimum local : Un point $\overrightarrow{x} \in \mathcal{X}$ est un minimum local du problème si et seulement si : $\forall \overrightarrow{x'} \in \mathcal{V}(\mathcal{X}),\ f(\overrightarrow{x}) \leq f(\overrightarrow{x'})$,où *V(X)* définit un « voisinage » de $\overrightarrow{x}$.

2.3 Problèmes d'optimisation multiobjectifs

La plupart des problèmes d'optimisation réels sont décrits à l'aide de plusieurs objectifs ou critères souvent contradictoires devant être optimisés simultanément. Alors que, pour les problèmes n'incluant qu'un seul objectif, l'optimum cherché est clairement défini, celui-ci reste à formaliser pour les problèmes d'optimisation multiobjectifs. En effet, pour un problème à deux objectifs contradictoires, la solution optimale cherchée est un ensemble de points correspondant aux meilleurs compromis possibles pour résoudre notre problème.

Prenons le cas d'une personne souhaitant acheter une voiture d'occasion. La voiture idéale est celle qui est peu chère avec peu de kilomètres, mais cette voiture idyllique n'existe pas (sinon je l'aurais achetée). Notre acheteur va donc devoir identifier les meilleurs compromis possibles correspondant à son budget (voir figure 1.2).

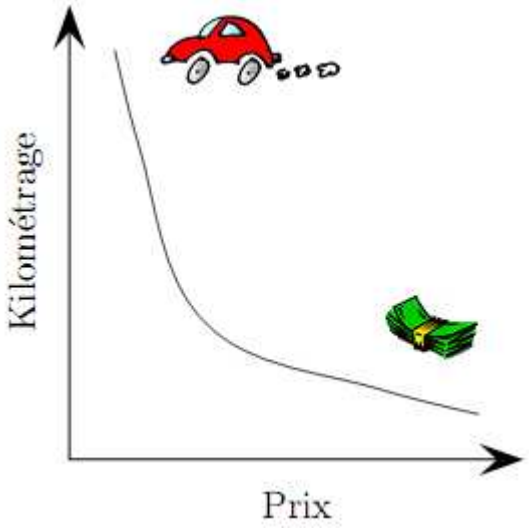


Fig.2.2. Relation entre kilométrage et prix.

2.3.1. Définitions :

2.3.1.1 Un problème :

Un problème dans le point de vue informatique (Computational Problem) veut dire comment faire relier un ensemble de données par un ensemble de résultats, où les données vont subir un ensemble d'opérations dont on les appelle le traitement. Donc il est conçu comme une relation $\prod \subseteq \ell \times S$ entre les entrées ou instances, et les sorties ou solutions.

Pour qu'une instance d'un problème soit compréhensible par un ordinateur numérique, elle doit être décrite comme une séquence finie de symboles d'un ensemble fini arbitraire appelé alphabet. De même, la solution d'une instance d'un tel problème est émise dans ce format. Généralement, l est infini alors que S peut être fini. [32]

2.3.1.2 Types des problèmes :

Les problèmes peuvent être classés selon les propriétés de l'ensemble des solutions :

- **Un problème de décision** : c'est un problème dont la réponse est tout simplement OUI ou NON.
- **Un problème polynomial réductible** : on a L1 et L2 deux problèmes de décision, on dit L1 est polynomial réductible à L2 s'il existe un algorithme polynomial qui convertit chaque instance d'entrée de L1 à une autre instance de L2.
- **Un problème de la classe P** : un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelque soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissances polynomiaux.

- **Un problème de la classe NP** : un problème de reconnaissance est dans la classe NP si, pour toute instance de ce problème, on peut vérifier, en un temps polynomial par rapport à la taille de l'instance, qu'une solution proposée ou devinée permet d'affirmer que la présence est « oui » pour cette instance. [33]
- **Un problème de la classe NP-hard** : on dit qu'un problème est dans la classe NP-hard si chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier.
- **Un problème de la classe NP-complet** : s'il appartient à NP et chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier. C'est un problème de décision et NP-hard qui cherche à trouver l'optimum.

Les problèmes d'optimisation multio-bjectifs sont une généralisation à n fonctions objectif des problèmes d'optimisation classiques (cf. Section 1.1). Ils sont définis formellement comme suit :

Considérons un problème de minimisation :
Soit $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$,
et soit $\mathcal{X}$ un ensemble fermé de $\mathbb{R}^n$
alors l'ensemble des minimiseurs est :

$$\hat{\mathcal{X}} = \left\{ \overrightarrow{x} \in \mathcal{X} \mid \forall \overrightarrow{x'} \in \mathcal{X}, \left(\exists i \in [1, \cdots, m] \; f_i(\overrightarrow{x}) < f_i(\overrightarrow{x'}) \right) \vee \left(\forall i \in [1, \cdots, m] \; f_i(\overrightarrow{x}) = f_i(\overrightarrow{x'}) \right) \right\}$$

et le minimum du problème est : $f(\hat{\mathcal{X}})$

D'après cette définition, il est clair que l'optimum n'est plus une simple valeur comme pour les problèmes à un objectif, mais un ensemble de points, appelé l'ensemble des meilleurs compromis ou le front Pareto. Dans la suite de ce mémoire nous adopterons la formulation suivante, équivalente à la précédente, mais plus souvent utilisée dans les travaux actuels :

Mininiser	$\overrightarrow{f}(\overrightarrow{x})$	(m fonctions à optimiser)
tel que	$\overrightarrow{g}(\overrightarrow{x}) \leq 0$	(q contraintes à satisfaire)
avec	$\overrightarrow{x} \in \mathbb{R}^n, \; \overrightarrow{f}(\overrightarrow{x}) \in \mathbb{R}^m, \; \overrightarrow{g}(\overrightarrow{x}) \in \mathbb{R}^q$	

2.3.1.4 Relations d'ordre et de dominance

Comme la solution optimale est une multitude de points de $\mathbb{R}^m$, il est vital pour identifier ces meilleurs compromis de définir une relation d'ordre entre ces éléments. Dans le cas des problèmes d'optimisation multiobjectifs, ces relations d'ordre sont appelées relations de

dominance. Plusieurs relations de dominance ont déjà été présentées : *la a-dominance* [34], *la dominance au sens de Geoffrion* [35], *la cone-dominance* [36], . . .
Mais la plus célèbre et la plus utilisée est ***la dominance au sens de Pareto***. C'est cette relation de dominance que nous allons définir et utiliser dans ce mémoire. De manière à définir clairement et formellement cette notion, les relations =, <= et < usuelles sont étendues aux vecteurs.

Définition 6 Soient u et v, deux vecteurs de même dimension,

$$\begin{array}{lll} \mathbf{u} = \mathbf{v} & ssi & \forall i \in \{1, 2, \cdots, m\},\ u_i = v_i \\ \mathbf{u} \leq \mathbf{v} & ssi & \forall i \in \{1, 2, \cdots, m\},\ u_i \leq v_i \\ \mathbf{u} < \mathbf{v} & ssi & \mathbf{u} \leq \mathbf{v} \wedge \mathbf{u} \neq \mathbf{v} \end{array}$$

Les relations >= et > sont définies de manière analogue.

Les relations définies précédemment ne couvrent pas tous les cas possibles. En effet, il est impossible de classer les points a = (1; 2) et b = (2; 1) à l'aide d'une de ces relations. Contrairement aux problèmes à un seul objectif où les relations usuelles <, <=, . . . suffisent pour comparer les points, elles sont insuffisantes pour comparer des points issus de problèmes multiobjectifs. Nous définissons donc maintenant la relation de dominance au sens de Pareto permettant de prendre en compte tous les cas de figures rencontrés lors de la comparaison de deux points (ici des vecteurs).

Définition 7 La dominance au sens de Pareto : Considérons un problème de minimisation. Soient u et v deux vecteurs de décision,

$$\begin{array}{rll} \mathbf{u} \prec \mathbf{v}\ (\mathbf{u}\ \textit{domine}\ \mathbf{v}) & ssi & \mathbf{f}(\mathbf{u}) < \mathbf{f}(\mathbf{v}) \\ \mathbf{u} \preceq \mathbf{v}\ (\mathbf{u}\ \textit{domine faiblement}\ \mathbf{v}) & ssi & \mathbf{f}(\mathbf{u}) \leq \mathbf{f}(\mathbf{v}) \\ \mathbf{u} \sim \mathbf{v}\ (\mathbf{u}\ \textit{est incomparable (ou non-dominé) avec}\ \mathbf{v}) & ssi & \mathbf{f}(\mathbf{u}) \lneq \mathbf{f}(\mathbf{v}) \wedge \mathbf{f}(\mathbf{v}) \lneq \mathbf{f}(\mathbf{u}) \end{array}$$

Pour un problème de maximisation, ces relations sont définies de manière symétrique.

Définition 8 Une solution $\mathbf{x} \in \mathcal{X}_f$ est dite non dominée par rapport à un ensemble $\mathcal{X}_a \subseteq \mathcal{X}_f$

si et seulement si : $\nexists \mathbf{x}_a \in \mathcal{X}_a,\ \mathbf{x}_a \prec \mathbf{x}$

Illustrons maintenant cette relation par un exemple en dimension 2 (cf. figure 2.3).

Sur cette figure, *F* (l'espace réalisable dans l'espace des objectifs) est l'image de X. Ainsi, chaque point $\mathbf{y^i}$ est l'image de $\mathbf{x^i}$ par $f : \mathbf{y^i} = f(\mathbf{x^i})$. Prenons le point $\mathbf{y^1}$ comme point de référence. Nous pouvons distinguer trois zones :

- La zone de préférence est la zone contenant les points dominés par $\mathbf{x^1}$,

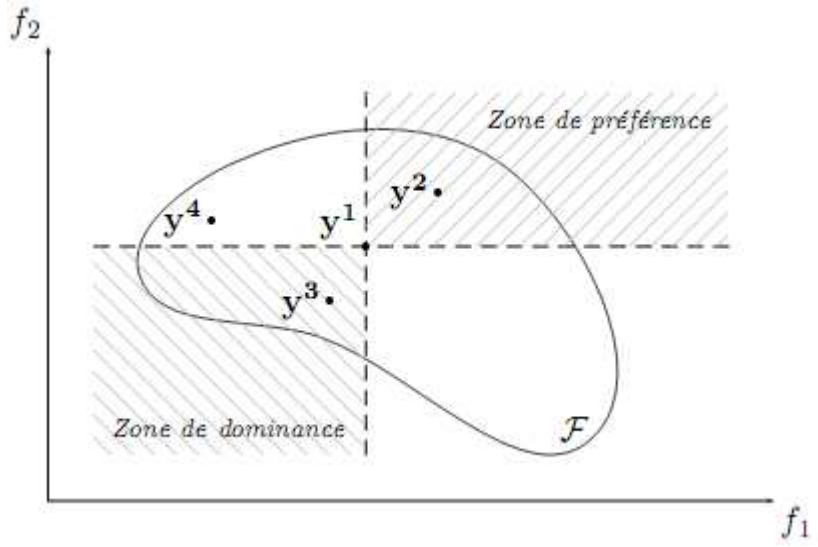


Fig. 2.3. Exemple de dominance.

- La zone de dominance est la zone contenant les points dominant $\mathbf{x^1}$,
- La zone d'incompatibilité contient les points incomparables avec $\mathbf{x^1}$.

Ainsi, il est clair que $\mathbf{x^2}$ est dominé par $\mathbf{x^1}$ ($\mathbf{x^1}$ est préféré à $\mathbf{x^2}$), que $\mathbf{x^3}$ domine $\mathbf{x^1}$ ($\mathbf{x^3}$ est préféré à $\mathbf{x^1}$), et que $\mathbf{x^4}$ est non dominé (incomparable) avec $\mathbf{x^1}$.
Forts de ce nouvel outil, nous pouvons maintenant définir l'optimalité dans le cas de problèmes multiobjectifs. Nous pouvons définir l'optimalité globale au sens de Pareto (c.à.d utilisant la dominance au sens de Pareto) :

Définition 9 L'optimalité globale au sens de Pareto : Un vecteur de décision $\mathbf{x} \in \mathcal{X}$ est dit Pareto globalement optimal si et seulement si : $\nexists \mathbf{y} \in \mathcal{X},\ \mathbf{y} \prec \mathbf{x}$. Dans ce cas, $\mathbf{f(x)} \in \mathcal{F}$ est appelé : **solution efficace.**

Nous pouvons maintenant définir le concept d'optimalité locale au sens de Pareto :

Définition 10 L'optimalité locale au sens de Pareto : Un vecteur de décision $\mathbf{x} \in \mathcal{X}$ est dit Pareto optimal localement si et seulement si, pour un $\delta > 0$ fixé : $\nexists y \in \mathcal{X},\ \mathbf{f}(\mathbf{y}) \in B(\mathbf{f}(\mathbf{x}), \delta)$ et $y \prec x$, où $B(\mathbf{f}(\mathbf{x}), \delta)$ représente une boule de centre **f (x)** et de rayon δ.

La figure 2.4 donne un exemple en dimension 2 d'optimalité locale. Le point f (x) est localement optimal, car il n'y a pas de point compris dans la boule B le dominant.

2.3.1.5 Front Pareto et surface de compromis

Nous rappelons que la solution que nous cherchons à un problème d'optimisation multiobjectif n'est pas un point unique mais un ensemble de points que nous avons appelé l'ensemble des meilleurs compromis à la Section 1.3.

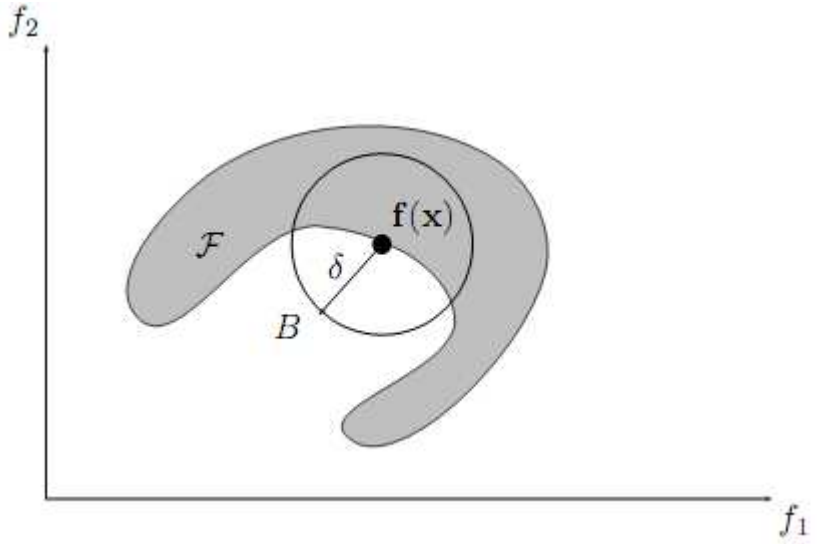


Fig.2.4. Exemple d'optimalité locale.

Nous avons défini jusqu'ici les notions de dominance (au sens de Pareto) et d'optimalité. Il nous reste maintenant à définir la solution d'un problème multiobjectif utilisant ces concepts. Cet ensemble des meilleurs compromis, appelé aussi surface de compromis ou front Pareto, est composé des points qui ne sont dominés par aucun autre. Nous définissons d'abord formellement l'ensemble des solutions non dominées :

Définition 11 Ensemble des solutions non dominées : Soit F l'image dans l'espace des objectifs de l'ensemble réalisable X. L'ensemble des solutions non dominées de X, est défini par l'ensemble ND(X) :

$$ND(\mathcal{X}) = \{\mathbf{x} \in \mathcal{X} \mid \mathbf{x} \textit{ est non dominé par rapport à } \mathcal{X}\}$$

Nous pouvons définir le front Pareto de F de manière analogue :

Définition 12 Front Pareto : Soit F l'image dans l'espace des objectifs de l'ensemble réalisable X. Le front Pareto ND(F) de F est défini comme suit :

$$ND(\mathcal{F}) = \{\mathbf{y} \in \mathcal{F} \mid \nexists \mathbf{z} \in \mathcal{F}, \mathbf{z} < \mathbf{y}\}$$

Le front Pareto est aussi appelé l'ensemble des solutions efficaces ou la surface de compromis Un exemple de surface de compromis (front Pareto) en dimension 2 est montré à a la figure 1.5. Dans cet exemple, le problème considéré est un problème de minimisation avec deux critères. Deux points particuliers apparaissent clairement : le point idéal et le point Nadir. Ces deux points sont calculés à partir du front Pareto. Le point idéal (resp. le point Nadir) domine (resp. est dominé par) tous les autres points de la surface de compromis.
Bien que ces points ne soient pas forcément compris dans la zone réalisable, ils servent souvent de pole d'attraction (resp. de répulsion) lors de la résolution du problème. Les coordonnées de ces deux points sont maintenant définies formellement.

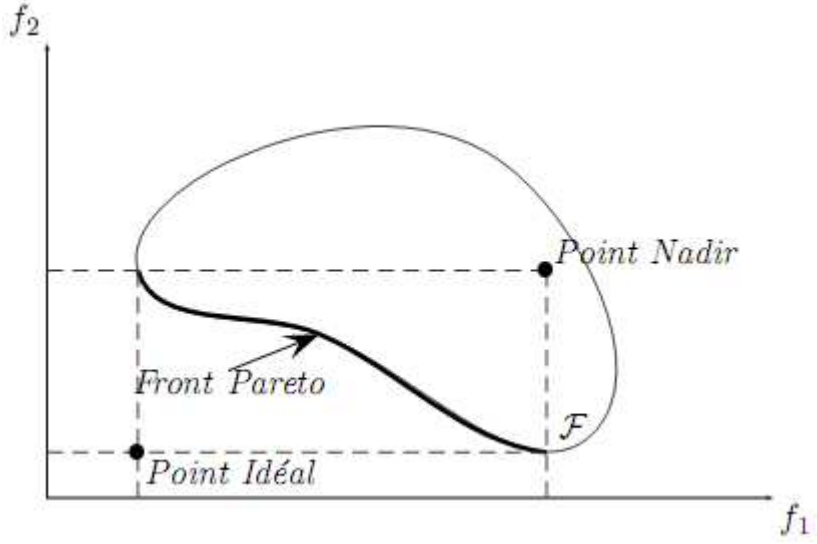


Fig. 2.5. Le front Pareto.

Définition 13 Point Idéal : Les coordonnées du point idéal (pi) correspondent aux meilleurs valeurs de chaque objectif des points du front Pareto. Les coordonnées de ce point correspondent aussi aux valeurs obtenues en optimisant chaque fonction objectif séparément.

$$pi_i = \min\{\mathbf{y}_i \mid \mathbf{y} \in ND(\mathcal{F})\}$$

Définition 14 Point Nadir : Les coordonnées du point Nadir (pn) correspondent aux pires valeurs de chaque objectif des points du front Pareto.

$$pn_i = \max\{\mathbf{y}_i \mid \mathbf{y} \in ND(\mathcal{F})\}$$

La figure 1.5 nous indique aussi que le front Pareto peut avoir des propriétés particulières quant à sa forme. La principale caractéristique utilisée pour comparer les formes de ces courbes est la convexité. Nous rappelons donc la définition de la convexité :

Définition 15 Convexité : Un ensemble A est convexe, si et seulement si l'équivalence suivante est vérifiée :

$$x \in A \wedge y \in A \Leftrightarrow Segment(x, y) \subset A$$

La convexité est le premier indicateur de la difficulté du problème. En effet, certaines méthodes sont dans l'incapacité de résoudre des problèmes non convexes de manière optimale. Mais il existe d'autres indicateurs tout aussi importants, notamment la continuité, la multi-modalité, la nature des variables de décision (entières ou réelles), . . .

2.4. Approches de résolution

Un grand nombre d'approches existent pour résoudre les problèmes multiobjectifs.
Certaines utilisent des connaissances du problème pour fixer des préférences sur les critères et ainsi contourner l'aspect multicritère du problème. D'autres mettent tous les critères au même niveau d'importance, mais là aussi il existe plusieurs façons de réaliser une telle opération.
Plusieurs ouvrages ou articles de synthèse ont été rédigés, des états de l'art plus complets peuvent être consultés notamment dans [37; 38 ; 35; 39 ; 40; 36].
Parmi toutes ces approches, il faut distinguer deux catégories : les approches non Pareto et les approches Pareto. Les approches non Pareto ne traitent pas le problème comme un véritable problème multiobjectif. Elles cherchent à ramener le problème initial à un ou plusieurs problèmes mono-objectifs. Par opposition aux méthodes non Pareto, les approches Pareto ne transforment pas les objectifs du problème, ceux-ci sont traités sans aucune distinction pendant la résolution.

Nous allons dans cette section, décrire les principales approches non Pareto et commenter leurs avantages et leurs inconvénients.

2.4.1 Les méthodes d'agrégation des objectifs

La première approche discutée ici est aussi la plus évidente. Elle consiste à transformer un problème multiobjectif en un problème à un objectif en agrégeant les différents critères sous la forme d'une somme pondérée :

$$\begin{array}{ll} \text{Minimiser} & f_\Sigma = \sum_{i=1}^{m} w_i \cdot f_i(\vec{x}) \\ \text{tel que} & \vec{g}(\vec{x}) \leq 0 \\ \text{avec} & \vec{x} \in \mathbb{R}^n,\ \vec{f}(\vec{x}) \in \mathbb{R}^m,\ \vec{g}(\vec{x}) \in \mathbb{R}^q \end{array}$$

Les w_i, appelés poids, peuvent être normalisés sans perte de généralité : $\sum w_i = 1$. Il est clair que la résolution d'un problème pour un vecteur de poids $\vec{w}$ fixée ne permet de calculer que quelques solutions Pareto optimales. Pour obtenir un ensemble contenant un grand nombre de solutions Pareto optimales, il faut résoudre plusieurs fois le problème en changeant à chaque fois les valeurs de $\vec{w}$.Cette approche a l'avantage évident de pouvoir réutiliser tous les algorithmes classiques dédiés aux problèmes d'optimisation à un seul objectif. C'est souvent la première approche adoptée lorsqu'un chercheur se retrouve devant un nouveau problème multiobjectif.Cependant cette approche a aussi deux inconvénients importants. Le premier est d^ u au fait que pour avoir un ensemble de points bien répartis sur le front Pareto, les différents vecteurs $\vec{w}$ doivent être choisis judicieusement. Il est donc nécessaire d'avoir une bonne connaissance du problème. Le deuxième inconvénient provient du fait que cette méthode ne permet pas, de calculer intégralement la surface de compromis lorsque celle-ci n'est pas convexe. La figure 2.6 illustre ce cas de figure en dimension 2. En effet, pour un vecteur de poids $\vec{w} = (w_1, w_2)$ fixé, la valeur optimale atteignable pour la fonction objectif créée est p. Les deux points Pareto optimaux trouvés sont A et C. En faisant varier le vecteur $\vec{w}$, il est possible de trouver d'autres points Pareto optimaux. Seulement, tous ces points se trouveront sur les parties convexes de la surface de compromis. Il n'existe pas de valeur possible pour $\vec{w}$ permettant de trouver par exemple le point B. En effet, cette approche ne permet pas d'approcher la totalité du front Pareto lorsque celui-ci est non convexe.

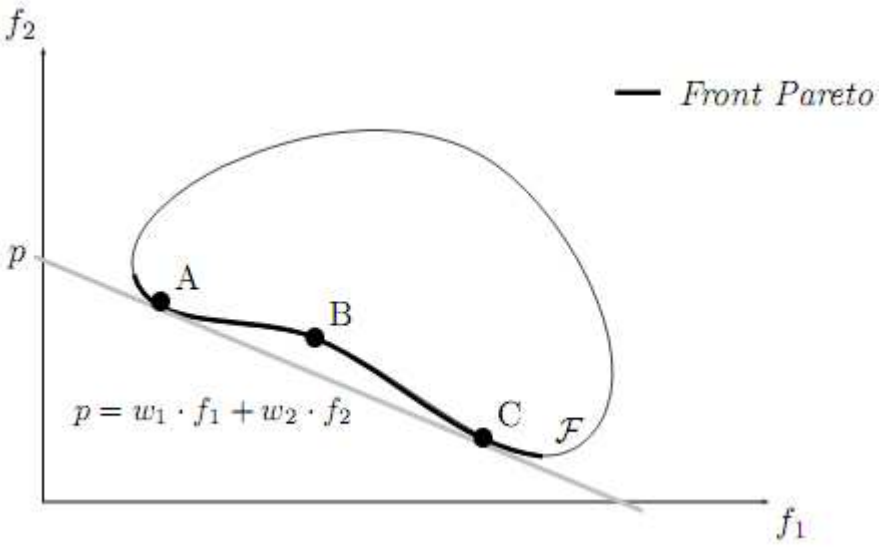


Fig. 2.6 Interprétation graphique de l'approche par pondération.

2.4.2 L'approche par $\epsilon-$ contrainte

Une autre façon de transformer un problème d'optimisation multiobjectif en un problème simple objectif est de convertir m-1 des m objectifs du problème en contraintes et d'optimiser séparément l'objectif restant. Le problème peut être reformulé de la manière suivante :

$$
\begin{array}{ll}
\text{Minimiser} & f_i(\overrightarrow{x}) \\
\text{tel que} & f_1(\overrightarrow{x}) \leq \epsilon_1 \\
 & \vdots \\
 & f_{i-1}(\overrightarrow{x}) \leq \epsilon_{i-1} \\
 & f_{i+1}(\overrightarrow{x}) \leq \epsilon_{i+1} \\
 & \vdots \\
 & f_m(\overrightarrow{x}) \leq \epsilon_m \\
\text{et que} & \overrightarrow{g}(\overrightarrow{x}) \leq 0 \\
\text{avec} & \overrightarrow{x} \in \mathbb{R}^n,\ \overrightarrow{f}(\overrightarrow{x}) \in \mathbb{R}^m,\ \overrightarrow{g}(\overrightarrow{x}) \in \mathbb{R}^q
\end{array}
$$

L'approche par $\epsilon-$contrainte doit aussi être appliquée plusieurs fois en faisant varier le vecteur $\overrightarrow{\epsilon}$ pour trouver un ensemble de points Pareto optimaux. Cette approche a l'avantage par rapport à la précédente de ne pas être trompée par les problèmes non convexes. Ainsi la figure 2.7 illustre, en dimension 2, le cas où un point $(\epsilon, f_{1_{min}})$, de la partie non convexe, est trouvé. La figure 2.7 montre aussi comment cette approche procède. En transformant des

fonctions objectifs en contraintes, elle diminue la zone réalisable par paliers. Ensuite, le processus d'optimisation trouve le point optimal sur l'objectif restant.

L'inconvénient de cette approche réside dans le fait qu'il faille lancer un grand nombre de fois le processus de résolution. De plus, pour obtenir des points intéressants et bien répartis sur la surface de compromis, le vecteur $\overrightarrow{\epsilon}$ doit être choisi judicieusement. Il est clair qu'une bonne connaissance du problème a priori est requise.

2.4.3 L'approche Min-Max

Cette approche consiste à transformer le problème multiobjectif en un problème à un seul objectif où l'on cherche à minimiser l'écart relatif par rapport à un point de référence appelé but, fixé par la méthode ou le décideur. Il existe plusieurs manières de caractériser la distance entre un point de référence (le but) et un autre, notamment à l'aide de normes.

Une norme est définie de la manière suivante :

$$L_r(\overrightarrow{f}(\overrightarrow{x})) = \left[\sum_{i=1}^{m} |B_i - f_i(\overrightarrow{x})|^r\right]^{\frac{1}{r}}$$

Les principales normes utilisées sont : $L_1 = \sum_{i=1}^{m} |B_i - f_i(\overrightarrow{x})|$, la distance classique, et la norme $L_\infty = \max_{i\in\{1,\cdots,m\}}(B_i - f_i(\overrightarrow{x}))$. C'est cette dernière qui est utilisée dans l'approche min-max appelée aussi approche de Tchebychev [38] :

$$\left|\begin{array}{ll} \text{Minimiser} & \max_{i\in\{1,\cdots,m\}}(B_i - f_i(\overrightarrow{x})) \\ \text{tel que} & \overrightarrow{g}(\overrightarrow{x}) \leq 0 \\ \text{avec} & \overrightarrow{x} \in \mathbb{R}^n,\ \overrightarrow{f}(\overrightarrow{x}) \in \mathbb{R}^m,\ \overrightarrow{g}(\overrightarrow{x}) \in \mathbb{R}^q \end{array}\right.$$

Dans cette approche, le point de référence joue un rôle fondamental. S'il est mal choisi, la recherche peut s'avérer être très laborieuse. La figure 1.8 illustre, en dimension 2, le cas d'une recherche avec un but B fixé. Il est clair que l'approche permet de traiter les problèmes non convexes à condition que le point de référence soit choisi judicieusement.

Les méthodes de résolution implémentant cette approche utilisent souvent le point idéal comme point de référence. Ce point idéal évolue donc en fonction de la recherche. En effet,

plus la surface de compromis courante trouvée par la méthode se rapproche du front Pareto optimal, plus le point idéal se rapprochera du point idéal du problème.

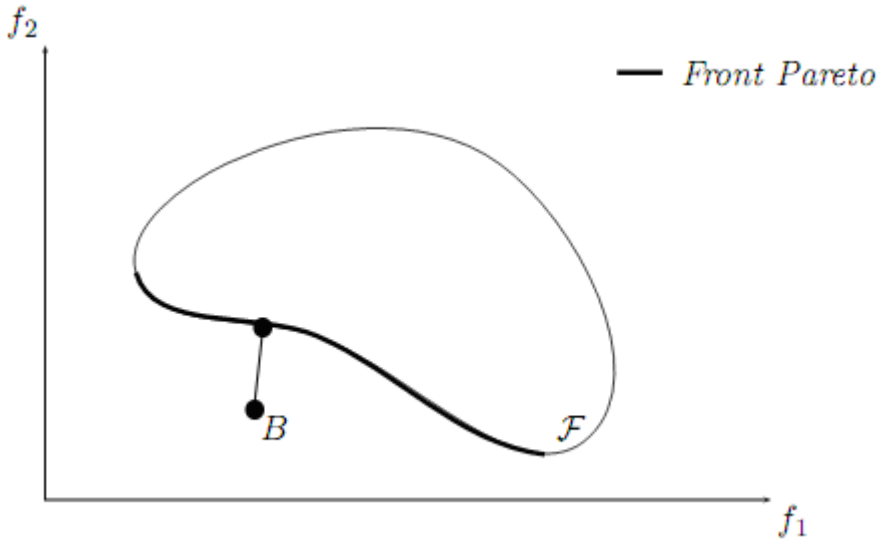


Fig 2.8. Interprétation graphique de l'approche Min-Max

2.4.4 Le but à atteindre :

Cette approche, comme celle de min-max, utilise un point de référence pour guider la recherche. Mais elle introduit aussi une direction de recherche, si bien que le processus de résolution devra suivre cette direction. A la différence de l'approche min-max, qui utilise des normes pour formaliser la distance au point de référence, l'approche du but à atteindre utilise des contraintes, à l'instar de l'approche ϵ-contrainte, pour déterminer la position du point de référence (aussi appel□ e le but). L'écart par rapport à ce but est contrôlé grâce à la variable λ introduite à cet effet :

$$
\begin{array}{|ll}
\text{Minimiser} & \lambda \\
\text{tel que} & f_1(\vec{x}) - w_1 \cdot \lambda \leq B_1 \\
 & \vdots \\
 & f_m(\vec{x}) - w_m \cdot \lambda \leq B_m \\
\text{et que} & \vec{g}(\vec{x}) \leq 0 \\
\text{avec} & \vec{x} \in \mathbb{R}^n,\ \vec{f}(\vec{x}) \in \mathbb{R}^m,\ \vec{g}(\vec{x}) \in \mathbb{R}^q
\end{array}
$$

Ainsi en minimisant λ et en vérifiant toutes les contraintes, la recherche va s'orienter vers le but B et s'arrêter sur le point A faisant partie de la surface de compromis (voir l'illustration en 2 dimensions à la figure 2.9). Cette approche permet, comme l'approche par ϵ-contrainte et l'approche min-max, de trouver les parties non convexes des fronts Pareto.

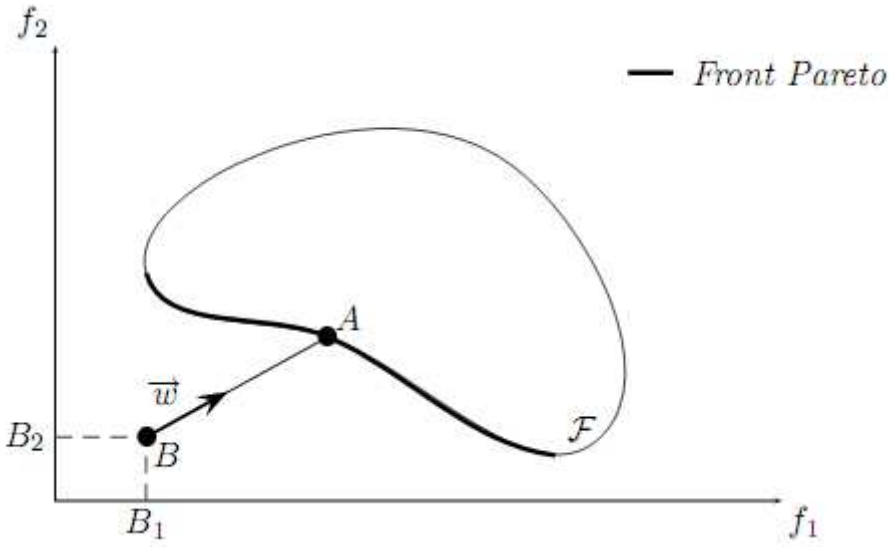


Fig. 2.9 Interprétation graphique de l'approche par "but à atteindre".

Cependant, cette approche, comme les précédentes, doit être itérée plusieurs fois dans le but d'obtenir un ensemble de points Pareto optimaux. Les paramètres ! w et B doivent être bien choisis par l'utilisateur. Bien que ces paramètres permettent une grande flexibilité de la recherche (orientation et but), s'ils sont mal choisis, ils peuvent, dans certains cas extrêmes, donner des résultats non cohérents.

Il est à noter que cette approche est très similaire à celle de la « goal programming » [41; 42;43], où les contraintes deviennent des égalités. Des variables d'écart sont alors introduites.

2.5. Discussion

Nous avons présenté dans ce chapitre les principaux concepts de l'optimisation multiobjectif. Pour ce chapitre et pour le reste de ce mémoire, nous nous plaçons dans le cadre de problèmes d'optimisation où il n'existe pas de modèle de préférences sur les critères (tous les critères sont de même importance). Toutes les problématiques liées à la modélisation des préférences, au choix de solutions au sein d'un ensemble de compromis, ou à la résolution interactive, sont traitées par la communauté d'aide à la décision dans un contexte multicritère. Le lecteur intéressé pourra consulter [44; 45] pour une présentation plus détaillée de ces problématiques.

Les différentes approches de résolution présentées dans ce chapitre sont des approches non Pareto. Elles transforment un problème d'optimisation multiobjectif en un ou plusieurs problèmes à un seul objectif. Que ce soit sous la forme d'une somme pondérée, ou sous la

forme d'une distance à un but, cette transformation permet d'utiliser facilement les méthodes d'optimisation issues de l'optimisation à un objectif.

Mais ces méthodes ont aussi des inconvénients. Certaines ne peuvent traiter complètement des problèmes non convexes et sont donc très sensibles à la forme du front Pareto (somme pondérée, . . .). Les autres, bien que pouvant traiter les problèmes non convexes, restent quand même sensibles à la forme du front Pareto (min-max, but à atteindre, . . .).

Un autre inconvénient important est qu'il faille relancer plusieurs fois les algorithmes de résolution avec des valeurs différentes pour certains paramètres (vecteur de poids par exemple) pour obtenir plusieurs points distincts de la surface de compromis. Ces méthodes nécessitent aussi souvent une bonne connaissance du problème a priori, notamment pour fixer les vecteurs de poids ou les points de référence.

Récemment les métaheuristiques, notamment les algorithmes évolutionnaires, ont permis la réalisation de méthodes de résolution dites Pareto très performantes. Elles permettent de déterminer en une seule exécution une approximation de l'intégralité du front Pareto, et ceci même si les problèmes sont non convexes. Le chapitre suivant est dédié à cette classe de méthodes relativement récentes.

Chapitre 3
Les métaheuristiques pour l'optimisation multiobjectif

Dans ce chapitre, nous présentons quelques méthodes de résolution Pareto fondées sur des métaheuristiques. Ainsi, nous introduisons les méthodes de recherche locale et les approches évolutionnaires. Nous décrivons aussi deux algorithmes évolutionnaires représentatifs adoptant une approche Pareto pour résoudre les problèmes d'optimisation multiobjectifs.

3.1. Introduction

Tout comme pour l'optimisation mono-objectif, on peut distinguer deux grandes familles de méthodes de résolution pour traiter un problème multiobjectif : les méthodes exactes et les méthodes approchées.

On remarque tout abord qu'il y a très peu de travaux sur les méthodes exactes dans le contexte de la résolution des problèmes d'optimisation multiobjectifs NP-difficiles, sans doute, à cause de la grande difficulté de ce type de problème. Les références [46; 47 ; 39] présentent la plupart des méthodes exactes existantes. Dans ce mémoire, nous nous intéressons à la résolution approchée de problèmes d'optimisation multiobjectifs NP-difficiles, notamment par la métaheuristique d'optimisation par colonie de fourmis. Ainsi, nous présentons brièvement dans ce chapitre deux types de métaheuristiques les plus connues : la recherche locale et les algorithmes évolutionnaires.

Pour une présentation plus élaborée des métaheuristiques, le lecteur est invité à consulter les références suivantes [48; 49; 50 ; 51; 52]. Les métaheuristiques ont été appliquées avec succès sur un grand nombre de problèmes académiques et réels : problème d'affectation quadratique, coloriage de graphe, voyageur de commerce, ... Le lecteur intéressé peut consulter, par exemple, [53] pour une présentation de quelques applications importantes.

3.2. Les méthodes de recherche locale

La première classe de métaheuristiques présentées regroupe les méthodes utilisant les principes de la recherche locale. Ces méthodes résolvent le problème d'optimisation de manière itérative. Elles font évoluer la configuration courante en la remplaçant par une autre issue de son voisinage, ce changement de configuration est couramment appelé un mouvement.

3.2.1 Déterminer le voisinage

Toutes les approches de recherche locale utilisent la notion de voisinage. Un aspect fondamental de ces approches est donc la détermination de ce voisinage.

Déterminer le voisinage consiste à caractériser tous ses éléments. Le voisinage est souvent représenté par une fonction N qui, à un point x, associe un ensemble de points N(x). Il existe une infinité de manières de choisir N, il faut adapter ce choix au problème, c'est-à-dire choisir la meilleure fonction N selon le problème considéré.

Définition 16 Fonction de voisinage : Soit X l'espace de recherche d'un problème. Une fonction de voisinage N est une association $\mathcal{N} : \mathcal{X} \longrightarrow 2^{\mathcal{X}}$, définissant, pour chaque point $\mathbf{x} \in \mathcal{X}$, un ensemble $\mathcal{N}(\mathbf{x}) \subseteq \mathcal{X}$ de points « proches » de x.

Un autre aspect important est la taille du voisinage. En effet, certaines fonctions N peuvent calculer un ensemble si grand qu'il est impossible de le traiter efficacement avec un ordinateur (dépassement de la taille mémoire par exemple). Il convient alors soit de changer la fonction N, soit, pour un voisinage complètement différent, soit, et c'est souvent la solution retenue, pour restreindre le voisinage à un ensemble contenant moins d'éléments.

Il est clair que plusieurs points x seront visités pendant la recherche. Certaines méthodes comme le recuit simulé ne calculent pas tout le voisinage mais uniquement certains points.

D'autres, comme la recherche Tabou, préconisent l'examen de tous les voisins pour réaliser un mouvement. Dans ce dernier cas de figure, il est important que le voisinage N(x) soit calculé rapidement. L'évaluation du ou des voisins est un autre aspect important de ces méthodes. En effet, le choix de la prochaine configuration dépend en grande partie de l'évaluation des voisins. Plus le voisinage est important et plus les phases d'évaluations sont longues. Il est

donc impératif que l'évaluation d'un candidat soit très rapide. Le plus souvent les méthodes de recherche locale utilisent des fonctions d'évaluation dites incrémentales. Ces fonctions se basent sur le changement local de x pour calculer de manière incrémentale la nouvelle évaluation. En effet, elles permettent, si un ordre de parcours du voisinage est respecté, d'évaluer chaque voisin très rapidement.

3.2.2 La descente

La descente est une méthode d'amélioration itérative simple permettant d'atteindre le premier optimum local. La descente pour un problème de minimisation peut être définie très simplement

Algorithme 1 Pseudo-code de la méthode de descente.

Paramètres d'entrée: $\mathcal{N}, f, \mathbf{x}_0$
$\mathbf{x}_{suiv} \leftarrow \mathbf{x}_0$
Répeter
    $\mathbf{x} \leftarrow \mathbf{x}_{suiv}$
    $\mathbf{x}_{suiv} \in \{\mathbf{x}' \mid \mathbf{x}' \in \mathcal{N}(\mathbf{x}) \wedge f(\mathbf{x}') = \min(\{f(\mathbf{y}) \mid \mathbf{y} \in \mathcal{N}(\mathbf{x})\})\}$
Jusqu'à $f(\mathbf{x}_{suiv}) > f(\mathbf{x})$
Renvoyer x

où N est la fonction de voisinage, f la fonction d'évaluation, et x0 la configuration initiale servant de point de départ à l'algorithme.

Ainsi le plus proche optimum local de x0 est trouvé. Mais celui-ci peut être loin de l'optimum global, et être une mauvaise approximation de cet optimum. Pour essayer de se rapprocher de l'optimum global, plusieurs techniques sont envisageables : la première technique couramment utilisée est celle de la relance. Elle consiste à recommencer une nouvelle recherche à partir d'un autre point initial, si possible loin du précédent. La deuxième technique est celle du chemin aléatoire. Elle consiste à effectuer, de temps en temps, un mouvement aléatoire pour diversifier la recherche et ainsi espérer se rapprocher de l'optimum global.

La descente est largement utilisée, et est souvent la première méthode expérimentée sur un nouveau problème. Elle permet, dans un temps de développement assez court, de bien appréhender le problème et de calculer rapidement des premières approximations de l'optimum global.

3.2.3 Le recuit simulé

Cette méthode de recherche a été proposée par des chercheurs d'IBM qui étudiaient les verres de spin. Ici, on utilise un processus métallurgique (le recuit) pour trouver un minimum. En effet, pour qu'un métal retrouve une structure proche du cristal parfait (l'état cristallin correspond au minimum d'énergie de la structure atomique du métal), on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement [54].

Ce processus métallurgique a été transposé à l'optimisation et a donné une méthode simple et efficace. Le pseudo code du recuit simulé est représenté dans l'algorithme 1.

Le fonctionnement de cet algorithme est le suivant :

- ✓ On commence par choisir un point de départ au hasard :
- ✓ On calcule un voisin de ce point (γ = Voisin (x)).
- ✓ On évalue ce point voisin et on calcule l'écart par rapport au point d'origine)) $(\Delta C = C(\gamma) - C(x))$.
- ✓ Si cet écart est négatif, on prend le point γ comme nouveau point de départ, s'il est positif on peut quand même accepter le point γ comme nouveau point de départ, mais avec une probabilité $e^{-\frac{\Delta C}{T}}$ (qui varie en sens inverse de la température T).
- ✓ Au fur et à mesure du déroulement de l'algorithme, on diminue la température $T(T = \alpha(T))$, souvent par paliers.
- ✓ On répète toutes ces étapes tant que le système n'est figé (par exemple, tant que la température n'a pas atteint un seuil minimal).

Une recherche de recuit simulé acceptera n'importe quelles nouvelles solutions qui sont évaluées comme des solutions supérieures, mais elle acceptera aussi les changements négatifs de qualité avec une probabilité qui dépend de la taille de diminution dans la qualité et la valeur courante de la température. La température commence à une haute valeur, idéalement assez haute que n'importe quelle solution inférieure aura presque une chance de 100 pourcent d'être acceptée, et les diminutions comme un processus exécute ces parcours. La chance

d'une solution négative pouvant être acceptée diminue, durant il y a une chance de zéro n'importe quel changement négatif de qualité peut être alloué. A ce point « refroidissement » est dit d'avoir lieu, et le système doit avoir convergé à la solution optimale. [55]

Nous donnons maintenant le pseudo-code d'un algorithme type du recuit simulé pour un problème de minimisation :

Algorithme 1 Recuit simulé.

Init T (température initial)
Init x (point de départ)
Init ΔT (temperature)
While (not (end))
  y = V*oisin* (x)
  $\Delta C = C\ (y)-C\ (x)$
  if $\Delta C < 0$ then $y = x$
  else if alea (0,1) < $e^{-\frac{\Delta C}{T}}$ then $y=x$
  $T=\alpha\ (T)$
  if $T < \Delta T$ then end (while)
Repeat (while)

Quelques applications du recuit simulé :

- ✓ Traitement d'images.
- ✓ Problèmes d'ordonnancement.
- ✓ Conception des circuits électroniques (Problème de placement et de routage).
- ✓ Organisation du réseau informatique du Loto (France).
- ✓ Collecte des ordures ménagères.
- ✓ Problème du voyageur de commerce (10000 villes). La longueur de la tournée excède de moins de 2% celle de la tournée optimale.

Avantages de la méthode

- ✓ Solution de bonne qualité.
- ✓ Méthode générale et facile à programmer.
- ✓ Souplesse d'emploi: De nouvelles contraintes peuvent être facilement incorporées

Inconvénients:

- ✓ Nombre important de paramètres.
- ✓ Temps de calcul: excessif dans certaines applications. [56]

3.2.4 La recherche Tabou

Cette méthode, mise au point par F. Glover [57] est conçue en vue de surmonter les minimax locaux de la fonction objective. C'est une technique d'optimisation combinatoire que certains présentent comme une alternative au recuit simulé.

A partir d'une configuration initiale quelconque, Tabou engendre une succession de configurations qui doivent aboutir à la configuration optimale. A chaque itération, le mécanisme de passage d'une configuration, soit x_n, à la suivante, soit x_{n+1}, est le suivant :

- ✓ on construit l'ensemble des « voisins » de x_n, c'est-à-dire l'ensemble des configurations accessibles en un seul « mouvement » élémentaire à partir de x_n (si cet ensemble est trop vaste, on en extrait aléatoirement un sous- ensemble de taille fixée): soit Voisinage (x_n) l'ensemble (ou le sous-ensemble) envisagé;
- ✓ on évalue la fonction objective f du problème de chacune des configurations appartenant à Voisinage (x_n). La configuration x_{n+1}, qui succède à la configuration x_n dans la chaîne de Markov construite par Tabou, est la configuration de Voisinage (x_n) en laquelle f prend sa valeur minimale.

Notons que la configuration x_{n+1} est adoptée même si $f(x_{n+1}) > f(x_n)$: c'est grâce à cette particularité que Tabou permet d'éviter les minimax locaux de f.

Cependant, telle quelle la procédure ne fonctionne généralement pas, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui provoque l'apparition d'un cycle. Pour éviter ce phénomène, on tient à jour, à chaque

itération, une « liste Tabou » de mouvements interdits ; cette liste -qui a donné son nom à la méthode- contient les mouvements inverses ($x_{n+1} \rightarrow x_n$) des m derniers mouvements ($x_n \rightarrow x_{n+1}$) effectués (typiquement m=7). La recherche du successeur de la configuration courante x_n est alors restreinte aux voisins de x_n qui peuvent être atteints sans utiliser un mouvement de la liste tabou. La procédure peut être stoppée dès que l'on effectué un nombre donné d'itérations, sans améliorer la meilleure solution atteinte jusqu'ici.

L'algorithme ainsi décrit est dit « Tabou simple ». Selon [58], il serait plus efficace que le recuit simulé pour le problème modèle du « coloration d'un graphe ». Cependant, le mode de construction de la liste tabou – qui, pour une simple raison d'économie de place mémoire, contient des mouvements interdits, et non des configurations interdites – peut bloquer l'accès à certaines solutions, pourtant non encore visitées. Pour éviter cet inconvénient, on peut employer la méthode plus complexe dite « tabou généralisée », qui prévoit la possibilité d'annuler le statut tabou d'un mouvement, lorsque le bénéfice escompté est « suffisant » (cette circonstance est appréciée à l'aide de la notion de « niveau d'aspiration ».

Le pseudo-code de cette méthode est représenté dans l'algorithme 3.

Algorithme 3 Recherche Tabou.

Init L=Ø (liste des points tabous)

Init x (point de départ)

Init k_{fin} (nb d'itérations max total)

Init k=0 et l=0

While (not (end))

y =V*oisinage* (x)–L (on prend un voisinage sans points tabous)

$\Delta C = C\ (y)\text{-}C\ (x)$

$x = y$ avec $y \in Voisinage\ (x) - L$ tel que min $C(y) - C(x)$

$L = L + \{x\}$

$k = k + 1$

Update L (on retire des mouvements tabous de L suivant des critères d'aspiration)

if $k = k_{fin}$ then end (while)

Repeat

3.2.5 Monté Carlo :

Les méthodes Monté Carlo consistent en des simulations expérimentales ou informatiques de problèmes mathématiques ou physiques, basées sur le tirage de nombres aléatoires. Généralement on utilise en fait des séries de nombres pseudo-aléatoires générées par des algorithmes spécialisés. Les propriétés de ces séries sont très proches de celles d'une véritable suite aléatoire.

La méthode Monté Carlo est également utilisée dans le domaine pharmaceutique : on génère in vitro de très nombreuses molécules aléatoires, puis on les passe au crible en testant leur effet sur tel ou tel cible. On repère ainsi des molécules intéressantes qui après une étude et une modification pourront donner naissance à de nouveaux médicaments.

L'orientation actuelle est même de réaliser la même chose in silico, c'est-à-dire de modéliser et de tester ces molécules dans un ordinateur. Le grand avantage de cette méthode est sa simplicité. Elle permet entre autres de visualiser l'effet de différents paramètres et de donner ainsi des orientations, d'étudier des structures intéressantes qui auraient été a priori écartées et

de trouver facilement des structures que l'on n'aurait pas aussi bien optimisées « à la main ». [55]

Les méthodes de types Monté Carlo recherchent l'optimum d'une fonction en générant une suite aléatoire de nombres en fonction d'une loi uniforme.

Algorithme :

- ✓ On génère un point initial x dans l'espace d'état, considéré comme solution courante.
- ✓ On génère aléatoirement un point x'.
- ✓ Si x' est meilleur que x alors x' devient la solution courante.
- ✓ Si le critère d'arrêt est satisfait alors fin sinon retour en à la deuxième étape [1]

3.2.6 Particle Swarm Optimization « PSO »

L'optimisation par particule d'essaim (PSO) c'est un algorithme d'intelligence d'essaim, inspiré par le social dynamique et le comportement émergeant qui surgissent dans les colonies socialement organisées, PSO est un algorithme basé sur la population, c'est-à-dire il exploite une population d'individus pour explorer les régions prometteuses de l'espace de recherche. Dans ce contexte, la population est nommée les essaims et les individus (c'est-à-dire les point de recherche) sont nommés les particules. Chaque particule meut avec une vitesse adaptable avec l'espace de recherche, et garde une mémoire de la meilleure position dont elle rencontre. Dans les variant global de PSO, la meilleure position qui est retenue par les individus de l'essaim est communiquée aux autres particules. Dans le variant local, chaque particule est assignée à un voisinage topologique consisté d'un nombre pré-spécifié de particules. Dans ce cas, la meilleure position retenue par les particules qui comprend le voisinage est communiquée entre eux. [59]

3.3 Les algorithmes évolutionnaires

La deuxième classe de métaheuristiques présentée dans ce mémoire est celle des algorithmes évolutionnaires [60]. On peut distinguer trois grandes classes d'algorithmes évolutionnaires : les algorithmes génétiques [61; 62], les stratégies d'évolution [63] et la programmation évolutive [64]. Ces méthodes se différencient par leur manière de représenter l'information et

par leur façon de faire évoluer la population d'une génération à l'autre. Un algorithme évolutionnaire est typiquement composé de trois éléments fondamentaux :

- une population constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné,
- un mécanisme d'évaluation des individus permettant de mesurer l'adaptation de l'individu à son environnement,
- un mécanisme d'évolution de la population permettant, grâce à des opérateurs prédéfinis, d'éliminer certains individus et d'en créer de nouveaux.

Parmi les composants d'un algorithme évolutionnaire, l'individu et la fonction d'évaluation correspondent respectivement à la notion de configuration et à la fonction d'évaluation dans les méthodes de voisinage. Le mécanisme d'évolution est composé de plusieurs opérateurs tels que la sélection, la mutation et le croisement.

La sélection a pour objectif de sélectionner des individus qui vont pouvoir se reproduire pour transmettre leurs caractéristiques à la génération suivante. Le croisement ou recombinaison est un opérateur permettant de construire des nouveaux individus enfants à partir des caractéristiques d'individus parents sélectionnés.

La mutation effectue des légères modifications de certains individus.

Comme exemple des algorithmes évolutionnaires, nous présentons maintenant les algorithmes génétiques.

3.3.1 Les algorithmes génétiques

Les théories de l'évolution montrent que les êtres vivants évoluent sous l'effet du milieu : les mieux adaptés ont plus de chances de survivre et donc de se reproduire. De génération en génération, les caractéristiques des individus les mieux adaptés on par conséquent plus de chances de se reproduire dans la population. La génétique, dont l'objet est d'étudier les mécanismes de l'hérédité, propose quant à elle un modèle permettant d'expliquer la transmission de ces caractéristiques d'une génération à l'autre.
Il existe en effet des entités responsables de la production des caractères héréditaires : ces entités sont appelées gènes et l'ensemble des gènes d'un individu définit son génotype. Par opposition, le phénotype d'un individu correspond à son apparence physique, ou plus

précisément à un ensemble de caractéristiques que l'on peut observer, mesurer ou qualifier chez lui (certaines pouvant nécessiter des investigations complexes, comme la mesure de taux de globules rouges dans un système sanguin) ; le phénotype est susceptible de varier au cours du temps par l'effet du milieu dans lequel il évolue, mais ces variations ne peuvent pas être transmises. Un gène est en fait un segment de chromosome, long filament d'ADN (acide désoxyribonucléique), qui est le matériel génétique de toutes les cellules. Les chromosomes sont situés dans le noyau des cellules. Toutes les cellules, sauf les cellules sexuelles, possèdent le même nombre de chromosomes, présents sous forme de paires : dans chaque paire, un des chromosomes vient du père et l'autre de la mère ; les chromosomes d'une même paire sont appelées homologues. Dans les cellules sexuelles, on n'observe que la moitié des chromosomes, chaque paire ne fournissant qu'un seul de ses chromosomes homologues.
Deux mécanismes permettent de fabriquer de nouvelles cellules. Le premier procède par division cellulaires : une cellule duplique son matériel génétique avant de se couper en deux copies conformes. Ou presque conformes : une erreur de reproduction peut se produire, affectant un gène ; il y a alors mutation de ce gène. Ce mécanisme est mis en œuvre dans la reproduction asexuée. Le second fait intervenir deux parents pour fabriquer un enfant : c'est celui qui intervient dans la « reproduction » sexuée, qu'il convient plutôt d'appeler procréation, puisque l'individu créé n'est pas une reproduction de ses parents. Dans ce mécanisme, les cellules sexuelles transmises par les parents apportent chacune la moitié des chromosomes de l'enfant, ce qui lui redonne bien le nombre voulu de chromosomes.
A défaut de pouvoir créer de nouveaux gènes, la procréation favorise un brassage génétique, puisque le patrimoine génétique de l'enfant est issu à l'égalité des deux parents (alors que la reproduction conforme n'évolue que du fait des mutations qui se produisent aléatoirement et rarement, du moins dans des conditions normales). Deux opérations viennent s'ajouter à ce brassage de la procréation pour accroître encore la diversité qui en résulte : les mutations, déjà évoquées, et le crossing-over (ou enjambement). Le crossing-over a été introduit pour expliquer comment des gènes situés sur un même chromosome (gènes liés) peuvent ne pas subir le même sort : l'un est transmis et l'autre non. Il arrive en effet, lors de la production des chromosomes qu'un parent va transmettre à l'enfant par l'intermédiaire des cellules sexuelles qu'il fabrique, qu'un échange s'opère, chez ces parent, entre chromosomes homologues : une séquence de l'un se substitue à une séquence de l'autre : cet échange s'appelle crossing-over. Le chromosome qui reçoit l'enfant est alors constitué de morceaux des deux chromosomes homologues détenus par la cellule parentale et provenant donc des grands-parents de cet

enfant. On remarquera que deux gènes proches sur un chromosome ont une probabilité plus faible d'être séparés par un crossing-over que deux gènes éloignés.
Les algorithmes génétiques vont s'inspirer, de façon plus ou moins fidèle, de ces mécanismes, sans pour autant en épuiser la complexité. Leurs prémices remontent au début des années soixante, lorsque J. H. Holland étudie les systèmes adaptatifs, dès 1962, à l'université du Michigan, mais l'expression « algorithmes génétiques » n'apparaît qu'en 1967. Vers 1975, J. H. Holland propose « la théorie des schémas », tentative de formalisation et d'explication théorique du comportement des algorithmes génétiques.
Sans doute suscitées par le temps de calcul important que nécessitant les algorithmes génétiques, les études sur le parallélisme de ces méthodes débutent aussi vers cette époque. Comme les algorithmes génétiques sont de gros consommateurs de temps, ce qui ne leur permet pas toujours d'atteindre des solutions convaincantes en des temps acceptables, il faut espérer que ces traitements en parallèle, pour lesquels les algorithmes génétiques semblent particulièrement adaptés, permettront de réduire les temps de calcul à des valeurs raisonnables et de rendre ainsi les algorithmes génétiques plus facilement applicables en pratique. [65]
La Figure 3.1 donne un exemple de représentation de l'information stocké dans un chromosome codé sur 8 bits. L'évaluation de cet individu consiste à transformer la chaîne 0/1 du chromosome en une valeur réelle, appelée valeur d'adaptation. La fonction d'évaluation dépend principalement de l'objectif du problème. Si cette fonction est uniquement basée sur la fonction objectif du problème, on utilisera le terme de fonction de coût pour la désigner.

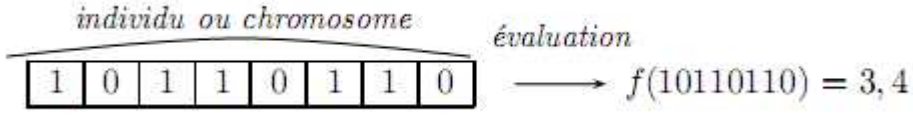


Fig. 3.1 Codage de l'information.

La valeur d'adaptation obtenue pour chaque chromosome lors de l'évaluation est utilisée, notamment lors des opérations de sélection, pour choisir les individus amener à se reproduire par des croisements ou à être utilisés par d'autres opérateurs génétiques.
Le croisement permet de produire deux nouveaux individus, appelés les enfants, à partir de deux individus, appelés les parents. La figure 3.2 montre un exemple de croisement monopoint, qui consiste à échanger les segments des deux parents déterminés par le point de croisement.

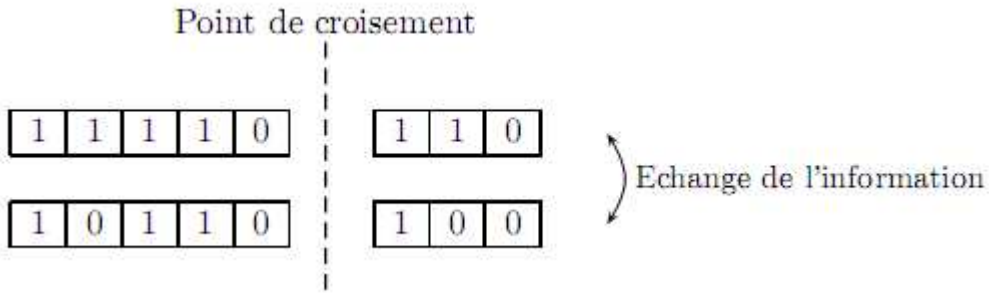


Fig. 3.2 Opérateur de croisement monopoint appliqué à deux chromosomes codés sur 8 bits.

La mutation consiste à changer la valeur de certaines variables du chromosome. Les variables à modifier sont le plus souvent choisies aléatoirement. La figure 3.3 montre un exemple de mutation sur un chromosome codé sur 8 bits.

Ainsi, la nouvelle population construite à partir des opérateurs génétiques présentés devient la population de référence de la prochaine génération. Ce cycle d'opérations continue tant que la méthode n'a pas rencontré une condition d'arrêt définie préalablement, un nombre maximal de générations par exemple.

Les algorithmes génétiques, et plus généralement les algorithmes évolutionnaires ont permis de résoudre un grand nombre de problèmes, notamment en optimisation. Dans le contexte multiobjectif, l'approche évolutionnaire offre, par le biais de la notion de population, des mécanismes pertinants pour approcher la solution optimale (front Pareto).

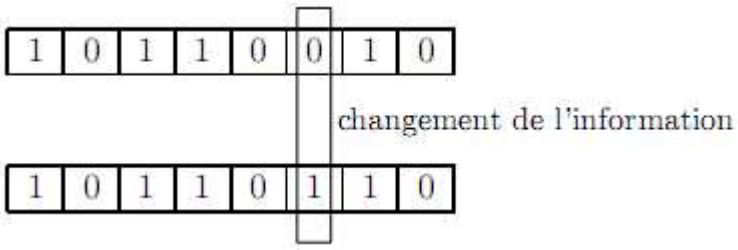


Fig. 3.3 Opérateur de mutation appliqué à deux chromosomes codés sur 8 bits.

Nous présentons maintenant trois de ces mécanismes : une sélection Pareto, l'élitisme et la diversification.

3.3.2 Un mécanisme de sélection Pareto

Un des principaux avantages des algorithmes évolutionnaires pour l'optimisation multiobjectif est qu'ils permettent non seulement la mise en oeuvre d'approches non Pareto (agrégation des objectifs, . . .), mais aussi l'implémentation d'approches Pareto. En effet, certains mécanismes de sélection implémentent la relation de dominance réalisant ainsi une sélection Pareto.

Une sélection Pareto utilise la relation de dominance pour affecter des rangs aux individus de la population, faisant apparaître la notion de front. Citons par exemple la technique de ranking, dont l'idée, suggérée dans [62], fut reprise et implémentée dans l'algorithme NSGA [66]. Cet algorithme, toujours très populaire dans la communauté EMOO (Evolutionary Multi Objective Optimization), sert encore de base pour le développement de nouveaux algorithmes.

Goldberg présenta dans ses travaux une procédure itérative de seulement 10 lignes pour calculer ce rang : initialement, tous les individus non dominés de la population reçoivent le rang 1 et sont retirés temporairement de la population. Puis, les nouveaux individus non dominés reçoivent le rang 2 avant d'être à leur tour retiré. Le processus s'itère tant qu'il reste des individus dans la population. La valeur d'adaptation de chaque individu correspond à son rang dans la population. Ainsi, l'évaluation d'un individu ne dépend pas uniquement de lui même, mais aussi de la population (cf. Figure 3.4).

Ce principe a l'avantage de ne pas hiérarchiser les objectifs entre eux. Mais l'augmentation de la taille de l'espace de recherche (surtout créée par l'augmentation du nombre de variables) peut influencer la performance de cette sélection. En effet, comment sélectionner certains individus si tous ont le même rang ! Ce cas peut arriver d'autant plus que le nombre de variables est grand, et que le problème est multimodal.

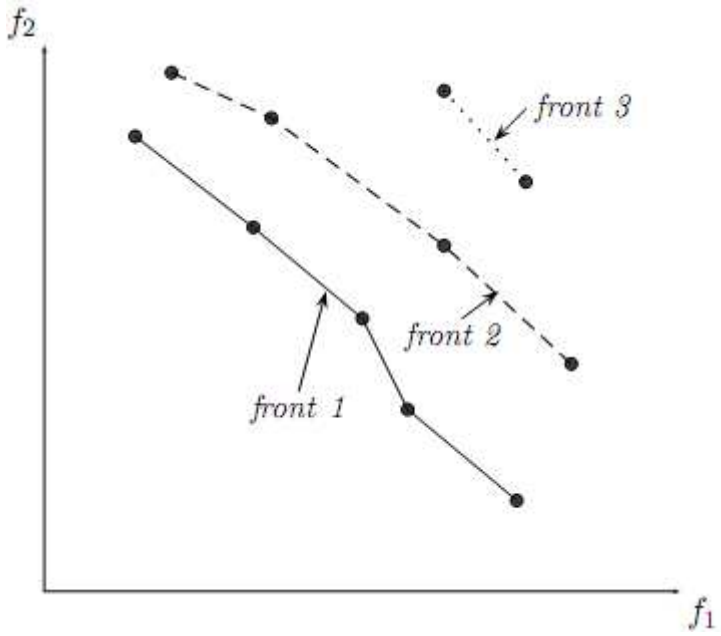


Fig. 3.4 Le ranking.

3.3.3 L'élitisme

L'élitisme permet de conserver les meilleurs individus dans les générations futures.

Une des premières implémentations de ce mécanisme dans un algorithme génétique est présentée dans [67].

L'élitisme est introduit pour conserver les bonnes solutions lors du passage de la génération courante à la prochaine génération. Conserver ces solutions pour les générations futures permet d'améliorer les performances des algorithmes sur certains problèmes. En effet, dans ses expérimentations, De Jong a observé qu'un algorithme génétique élitiste améliorait significativement les résultats sur des fonctions unimodales. D'un autre coté, l'élitisme peut causer une convergence prématurée sur des fonctions multimodales.

Réaliser un algorithme élitiste dans le cadre des problèmes multiobjectifs est plus difficile que pour les problèmes à un objectif. En effet, la meilleure solution n'est plus un unique individu, mais tout un ensemble dont la taille peut aller jusqu'à dépasser la taille maximale de la population. Deux adaptations du mécanisme élitiste sont considérées :

la première approche regroupe les algorithmes, fondés sur les travaux de De Jong, qui conservent pour les générations futures les k meilleurs individus [68; 69]. Mais comment avec cette approche sélectionner k individus, si l'ensemble des points non dominés actuel comporte

plus de k solutions? Il y a un risque de perdre une partie du front Pareto optimal, et le concept de l'élitisme n'est plus complètement présent.

Les approches récentes [70; 71 ; 72] tendent à utiliser une population externe d'individus dans laquelle est stocké le meilleur ensemble des points non dominés découverts jusqu'ici. Cet ensemble est mis à jour continuellement pendant la recherche, et les individus stockés continus à pouvoir être choisis par l'opérateur de sélection. Ils peuvent ainsi se reproduire et transmettre leurs caractéristiques aux générations suivantes.

Actuellement, les algorithmes élitistes obtiennent de meilleurs résultats sur un grand nombre de problèmes multiobjectifs [72; 73].

3.3.4 Maintenir la diversité :

La diversité est une notion déjà importante lors de l'optimisation de problèmes à un objectif, elle devient prépondérante lorsque l'on traite de problèmes multiobjectifs.
Maintenir un certain degré de diversité dans la population d'un algorithme évolutionnaire consiste à éviter que la population ne converge prématurément vers une petite zone de l'espace de recherche ou de l'espace des objectifs. En effet, si il n'existe pas de mécanisme de contrôle de la diversité, les opérations de sélection vont privilégier trop vite certains individus meilleurs à cette étape de la recherche. Cette convergence prématurée a comme effet de limiter la recherche à un sous-ensemble plus restreint de l'espace de recherche, qui peut ne contenir aucune solution optimale. Dans le cas de problèmes multiobjectifs, converger prématurément rend impossible la découverte de l'intégralité du front Pareto.
En effet, les individus de la population se focaliseront sur une partie du front Pareto, et ne se répartiront pas sur la totalité de ce front Pareto.
Pour remédier à ce problème, de nombreuses techniques ont été développées. Celles-ci influent sur la pression de sélection, afin de privilégier certains individus, permettant d'obtenir une population plus diversifiée. Cependant, ces techniques ajoutent un coût calculatoire non négligeable pour l'algorithme, elles doivent donc être choisies avec soin.
Nous allons maintenant présenter les mécanismes de diversification les plus couramment intégrés aux algorithmes évolutionnaires.
Le sharing

Le sharing consiste à modifier la valeur de coût d'un individu (calculée uniquement à partir de la fonction objectif du problème). C'est cette nouvelle valeur qui sera utilisée comme valeur d'adaptation par l'opérateur de sélection. Cette technique, introduite dans [73], est largement utilisée aujourd'hui.

Pour éviter qu'un trop grand nombre d'individus ne se concentrent autour d'un même point, il faut pénaliser la valeur d'adaptation en fonction du nombre d'individus au voisinage du regroupement : plus les individus sont regroupés, plus leur valeur d'adaptation est faible, et des individus proches les uns des autres doivent partager leur valeur d'adaptation. Dans la pratique, on estime ce taux de concentration en ouvrant un domaine autour d'un individu, puis on calcule les distances entre les individus contenus dans ce domaine.

Pour déterminer les bornes du domaine ouvert autour de l'individu choisi, on définit une distance maximale, appelée σ_{share}, au delà de laquelle les individus ne seront plus considérés comme faisant parti du domaine ouvert. La distance séparant deux individus

i et j est calculée grâce à la fonction d(i; j). La valeur d'adaptation F(i) d'un individu

$i \in P(population)$ est égale à son coût *F'(i)* divisé par sa valeur de niche :

$$F(i) = \frac{F'(i)}{\sum_{j \in P} Sh(d(i,j))}$$

Où la fonction *Sh* et définie comme suit :

$$Sh(d(i,j)) = \begin{cases} 1 - \left(\frac{d(i,j)}{\sigma_{share}}\right)^2 & \text{si } d(i,j) < \sigma_{share} \\ 0 & \text{sinon} \end{cases}$$

La fonction *d(i; j)* de calcule de distance peut être définie dans l'espace de recherche, par exemple à l'aide d'une distance de Hamming, ou dans l'espace objectif. Ce choix dépend souvent du problème, car le maintient de la diversité dans l'espace objectif, bien qu'il soit souvent plus simple à réaliser, n'assure pas forcément le maintient de la diversité dans l'espace de recherche.

La réinitialisation

La réinitialisation est une technique largement utilisée par toutes les métaheuristiques, les algorithmes évolutionnaires n'échappant pas à la règle. Lors d'approches par population, elle consiste à réinitialiser un certain nombre d'individus de la population, par exemple de manière aléatoire. En introduisant de manière régulière certains individus générés aléatoirement, de nouvelles zones de l'espace de recherche, peuvent être inexplorées jusqu'ici, peuvent être découvertes.

Le Crowding

L'approche par « Crowding » consiste à déterminer un représentant par niche découverte.

A la différence du « sharing », où tous les individus sont susceptibles d'être sélectionnés et de participer aux phases de croisement, mutation et sélection, avec le « crowding », seuls les représentants participeront aux différentes étapes de l'algorithme. Le « Crowding » fut introduit par De Jong [67] et fut adapté notamment au travers des travaux de [74].

3.3.5 Les méthodes hybrides

Une autre façon d'améliorer les performances d'un algorithme ou de combler certaines de ses lacunes consiste à le combiner avec une autre méthode [75]. Ce principe général, appelé hybridation, peut s'appliquer pour un grand nombre de méthodes. Les algorithmes évolutionnaires ne font pas exception à la règle, et une multitude d'algorithmes hybrides ont fait leur apparition ces dernières années. Un cas particulier de l'hybridation entre deux méthodes consiste à combiner un algorithme génétique et une méthode de recherche locale.Dans une telle hybridation, on substitue souvent la mutation par une méthode de recherche locale. En effet, les méthodes de recherche locale remplacent la configuration courante par une autre voisine. Il y a donc peu de modifications séparant les deux configurations. L'opérateur de mutation des algorithmes évolutionnaires effectue lui aussi des modifications légères sur la configuration sélectionnée. De manière intuitive, un grand nombre de chercheurs ont donc substitué une méthode de recherche locale à l'opérateur de mutation, pour créer une méthode hybride. Nous pouvons ainsi citer pour le cas des problèmes multiobjectifs: la méthode MOTS [76; 77] combinant une population et une recherche Tabou, la méthode PSA [78] combinant un algorithme génétique et le recuit simulé, la méthode M-PAES [79] intégrant un schéma généralisant l'implémentation d'un grand nombre d'algorithmes hybrides pour l'optimisation des problèmes multiobjectifs.

3.4 Quelques algorithmes évolutionnaires performants

Récemment, beaucoup de recherches ont été menées sur l'application des algorithmes évolutionnaires aux problèmes d'optimisation multiobjectifs. Celles-ci ont permis de mettre en avant l'intérêt d'utiliser des méthodes d'optimisation basées sur le concept de population. Nous présentons maintenant deux algorithmes évolutionnaires représentatifs, résolvant des problèmes d'optimisation multiobjectifs. Ces deux méthodes ont donné lieu à des travaux récents et nombreux [72; 80], illustrant leurs originalités et leurs bonnes performances sur de nombreuses instances de problèmes.

3.4.1 Le 'Non Sorting Dominated Genetic Algorithm II' (NSGA-II)

Srinivas et Deb en 1994 [66] ont réalisé une implémentation quasi directe du schéma de sélection introduit par Goldberg [62], nommé le Non Sorting Genetic Algorithm (NSGA). Les différents fronts de compromis de la population sont recueillis un par un, et la valeur d'adaptation est calculée sur chaque front de manière indépendante, préservant ainsi la diversité. Une nouvelle version élitiste de cet algorithme, nommée Non Sorting Genetic Algorithm II (NSGA-II), a été présentée dans [69]. NSGA-II intègre un opérateur de sélection, basé sur un calcul de la distance de « crowding », très différent de celui de NSGA. Comparativement à NSGA , NSGA-II obtient de meilleurs résultats sur toutes les instances présentées dans les travaux de K. Deb, ce qui fait de cet algorithme un des plus utilisés aujourd'hui.

Fonctionnement général

NSGA-II est un algorithme élitiste n'utilisant pas d'archive externe pour stocker l'élite.
Pour gérer l'élitisme, NSGA-II assure qu'à chaque nouvelle génération, les meilleurs individus rencontrés soient conservés.

Pour comprendre le fonctionnement de l'algorithme, plaçons nous à la génération t.
Comme le montre la Figure 3.5, deux populations (P_t et Q_t de taille N) coexistent. La population P_t contient les meilleurs individus rencontrés jusqu'à la génération t, et la population Q_t est formée d'individus autres, issus des phases précédentes de l'algorithme. La première étape consiste à créer la population $R_t = P_t \bigcup Q_t$ et à appliquer une procédure de ranking pour identifier les différents fronts F_i de solutions non dominées (les meilleurs individus se retrouvent donc dans le, ou les, tous premiers fronts).

La deuxième phase consiste à construire une nouvelle population P_{t+1} contenant les N meilleurs individus de R_t. Il faut pour cela inclure intégralement les meilleurs fronts F_i (c'est

à-dire en commençant à l'indice 1) tant que le nombre d'individus présents dans P_{t+1} est inférieur à N. Il reste donc à ce stade $N - |P_{t+1}|$ individus à inclure dans P_{t+1}.

Pour cela, une procédure de crowding est appliquée sur le premier front F_i non inclus. Les $N - |P_{t+1}|$ meilleurs individus au sens de cette procédure de crowding sont insérés dans P_{t+1}.

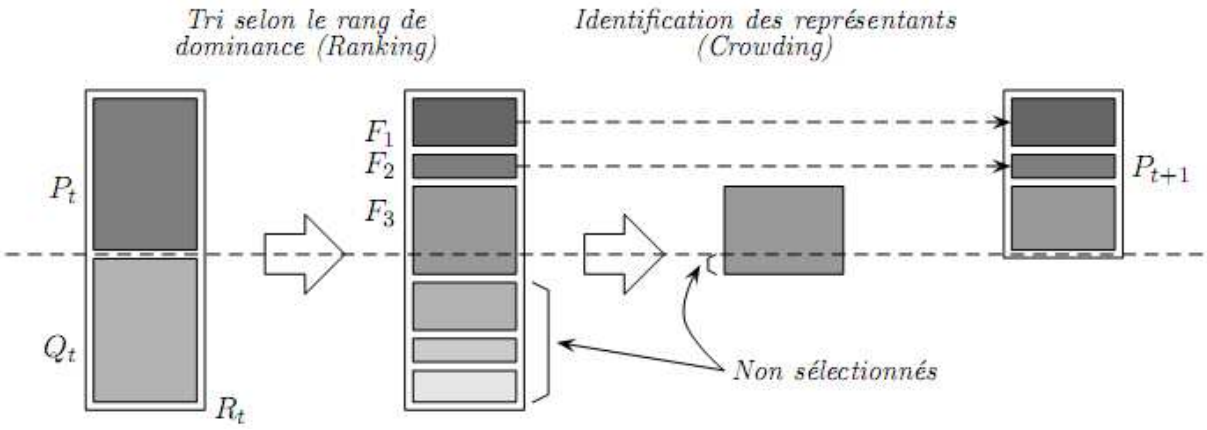


Fig. 3.5 Fonctionnement de NSGA-II : schéma extrait de [40].

La troisième phase consiste à remplir la population Q_{t+1}. Il suffit alors d'utiliser les opérateurs de sélection, croisement et mutation sur les individus de P_{t+1}, puis d'insérer les descendants dans Q_{t+1}. L'algorithme 3 reprend toutes les étapes décrites ci-dessus.

Algorithme 3 Pseudo-code de l'algorithme *NSGA-II*.

Initialiser les populations P_0 et Q_0 de taille N
`Tant que` *critère_d'arrêt_non_rencontré* `faire`
- Création de $R_t = P_t \bigcup Q_t$
- Calcul des différents fronts F_i de la population R_t par un algorithme de "ranking"
- Mettre $P_{t+1} = \emptyset$ et $i = 0$,
- `Tant que` $|P_{t+1}| + |F_i| < N$ `faire`
  $P_{t+1} = P_t \bigcup F_i$
  $i = i + 1$
  `FinTantQue`
- Inclure dans P_{t+1} les $(N - |P_{t+1}|)$ individus de F_i les mieux répartis au sens de la distance de "crowding"
- Sélection dans P_{t+1} et création de Q_{t+1} par application des opérateurs de croisement et mutation

`FinTantQue`

Calcul de la distance de « crowding »

Le calcul de valeur d'adaptation pour NSGA-II décrit dans [69] ne sert pas uniquement pour la sélection des opérateurs de croisement et de mutation, mais intervient aussi dans la sélection des individus à inclure dans Pt+1 (la population contenant les élites).
C'est donc une phase importante pour laquelle les auteurs de NSGA-II ont développé une méthode particulière : la distance de crowding.

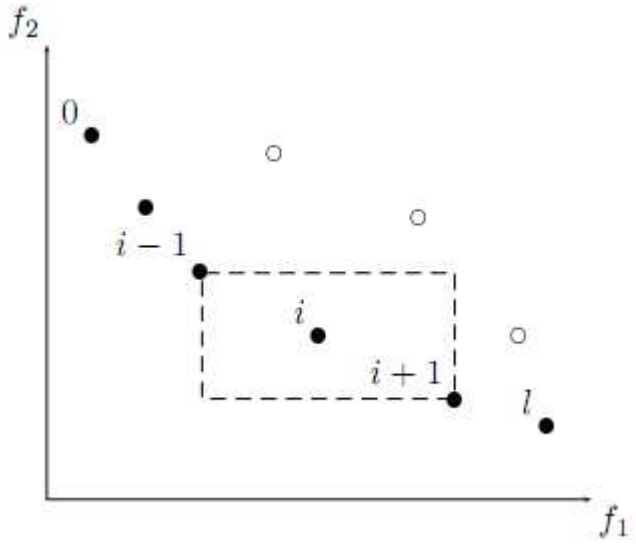


Fig. 3.6 Calcul d'un représentant (crowding) [40].

La distance de crowding di d'un point particulier i se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de i sur chaque objectif.
Sur la figure 3.6 est représenté l'hypercube en deux dimensions associé au point i. Un algorithme de calcul de la distance de crowding est détaillé dans [Deb, 2001]. Cet algorithme est de complexité $\mathcal{O}(MN\log(N))$, où M est le nombre d'objectifs du problème et N le nombre d'individus à traiter. Une fois tous les di calculés, il ne reste plus qu'à les trier par ordre décroissant et à sélectionner les individus possédant la plus grande valeur de crowding.
Les techniques de sélection en vue d'un croisement ou d'une mutation peuvent aussi bénéficier de ce type de calcul. En effet, l'opérateur de sélection le plus fidèle à l'esprit de Pareto tombe en défaut dés que les deux individus en compétition ont le même rang de dominance (i.e. sont non dominés). Pour pallier à ce problème, NSGA-II utilise la distance de « crowding » pour départager les deux individus. Ce processus nommé Crowded Tournament Selection permet de conserver les bonnes propriétés de la sélection Pareto tout en préservant une certaine diversité sur les individus sélectionnés.

3.4.2 Le “Strength Pareto Evolutionary Algorithm” (SPEA)

La méthode SPEA implémentée par Zitzler et Thiele [Zitzler and Thiele, 1998b] est l'illustration même d'un algorithme évolutionnaire élitiste. Pour réaliser cet élitisme, SPEA maintient une archive externe contenant le meilleur front de compromis rencontré durant la recherche.

Fonctionnement général

La première étape consiste à créer une population initiale (constituée par exemple d'individus générés aléatoirement). L'archive externe, au départ initialisée à l'ensemble vide, est mise à jour régulièrement en fonction des individus non dominés de la population.

A chaque itération de l'algorithme, on retrouve les étapes classiques sélection + croisement + mutation. Puis les nouveaux individus non dominés découverts viennent s'ajouter à l'archive, et les individus de l'archive dominés par le nouvel arrivant sont supprimés. Si l'archive vient à excéder une certaine taille (fixée au départ), alors une phase de clustering est appliquée dans le but de garder les meilleurs représentants.

La figure 3.7 (extraite de la thèse de Zitzler [Zitzler, 1999]) et l'algorithme 4 illustrent le schéma général de fonctionnement de l'algorithme.

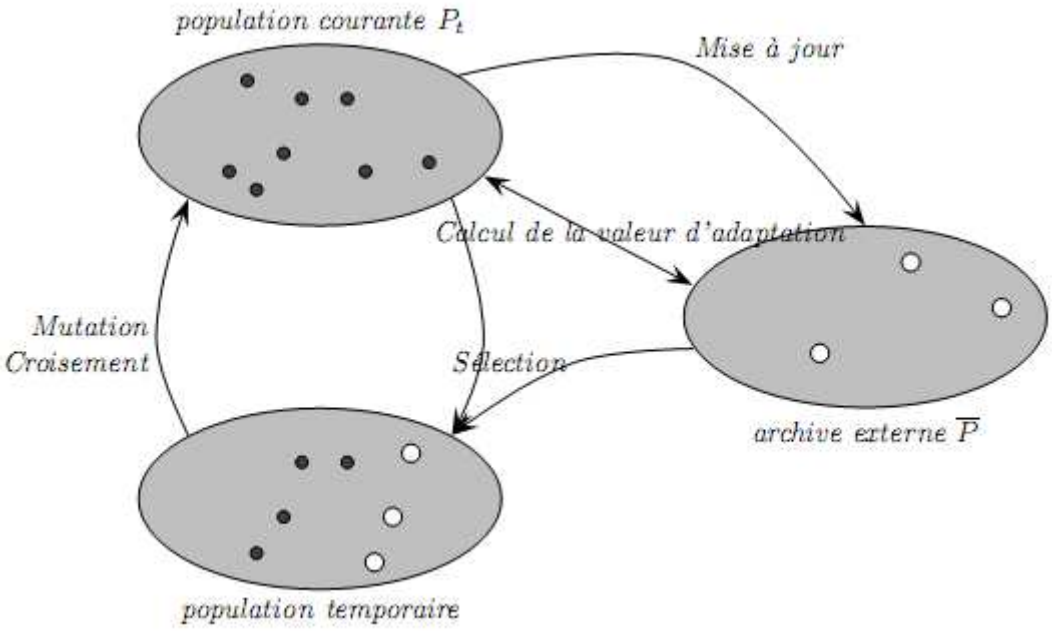


Fig. 3.7 Fonctionnement général de l'algorithme SPEA.

Algorithme 4 Pseudo-code de l'algorithme *SPEA*.

Initialiser la population P_0 et créer l'archive externe vide $\overline{P} = \emptyset$
Mise à jour de $\overline{P}$ à partir des individus non dominés de P_0
`Tant que` *critère_d'arrêt_non_rencontré* `faire`
- Calcul de la valeur d'adaptation pour tous les individus de $P + \overline{P}$
- Sélection dans $P_t + \overline{P}$ en fonction de la valeur d'adaptation
- Croisement
- Mutation
- Mise à jour de $\overline{P}$ à partir des individus non dominés de P

`FinTantQue`

Calcul de la valeur d'adaptation

Le calcul de la valeur d'adaptation de SPEA permet de diminuer les chances de sélection des individus ayant une moins bonne évaluation, ou des individus issus de niches déjà importantes. A la différence des techniques classiques de « sharing » qui sont basées sur la notion de distance euclidienne dans l'espace des objectifs, SPEA utilise la notion de dominance pour détecter les niches d'individus. De plus, de par la nature élitiste de l'algorithme, les individus de l'archive externe participent eux aussi à la sélection, le calcul d'adaptation doit donc les prendre en compte.

Ce calcul s'effectue en deux étapes. La première étape consiste à affecter une valeur, appelée valeur de dureté (S), aux individus de l'archive $\overline{P}$ externe. Cette valeur est déduite à partir du nombre d'individus qu'un élément $i \in \overline{P}$ domine faiblement dans la population courante P_t

$$S(i) = \frac{|\{j \mid j \in P_t \wedge f(i) \leq f(j)\}|}{|P_t| + 1}$$

La valeur d'adaptation d'un individu $i \in \overline{P}$ est égale à sa valeur de dureté : $F(i) = S(i)$

Pour un individu *j* de la population courante P_t, la valeur d'adaptation est calculée en réalisant la somme des valeurs de dureté des individus de $\overline{P}$ dominant *j* :

$$F(j) = 1 + \sum_{i \in \overline{P} \wedge f(i) \leq f(j)} S(i)$$

Le chiffre 1 est ajouté au total de la somme pour éviter que des individus de $\overline{P}$ aient une valeur d'adaptation plus grande que certains individus de P_t. Il est à noter que, dans ce calcul,

une plus petite valeur d'adaptation conduit à une plus grande chance de sélection de l'individu. Ainsi, plus un individu est dominé par les individus de $\overline{P}$ et plus sa valeur d'adaptation décroît, diminuant donc ses chances d'être sélectionné. Un exemple de calcul de valeurs d'adaptation est illustré à la figure 3.8. En observant cette figure, il est clair que l'individu évalué à 19/6 (qui est le résultat de la somme : 1+3/6+2/6+3/6+3/6+2/6) à une probabilité moindre d'être sélectionné que les individus évalués à 14/6.

Réduction par clusteringLorsque le nombre d'individus de l'archive externe est grand, les performances de l'algorithme peuvent se dégrader significativement. En effet, le nombre d'individus influe sur le calcul de la valeur d'adaptation, celui-ci devient moins able, et peut tromper la recherche en la focalisant, par exemple, sur des zones déjà explorées.Pour pallier à ce défaut, une solution consiste à utiliser des techniques de « clustering ». Ces techniques ont été étudiées intensivement dans le contexte des analyses de cluster [Morse, 1980] et ont été appliquées avec succès pour déterminer des partitions d'une collection relativement hétérogène d'éléments. La technique de « clustering » utilisée par SPEA est assez intuitive. Au début de la procédure, chaque individu constitue son propre groupe,puis on fusionne deux à deux les groupes les plus proches en terme de distance. Cette étape est itérée jusqu'à l'obtention du nombre désiré de groupes. Une fois les groupes identifiés, il ne reste plus qu'à choisir un représentant par groupe. Ce représentant peut être déterminé de plusieurs façons, par exemple en prenant le barycentre du groupe. C'est ce représentant qui sera gardé, les autres éléments étant tout simplement supprimés. Les étapes de la méthode de clustering sont illustrées à la figure 3.9.

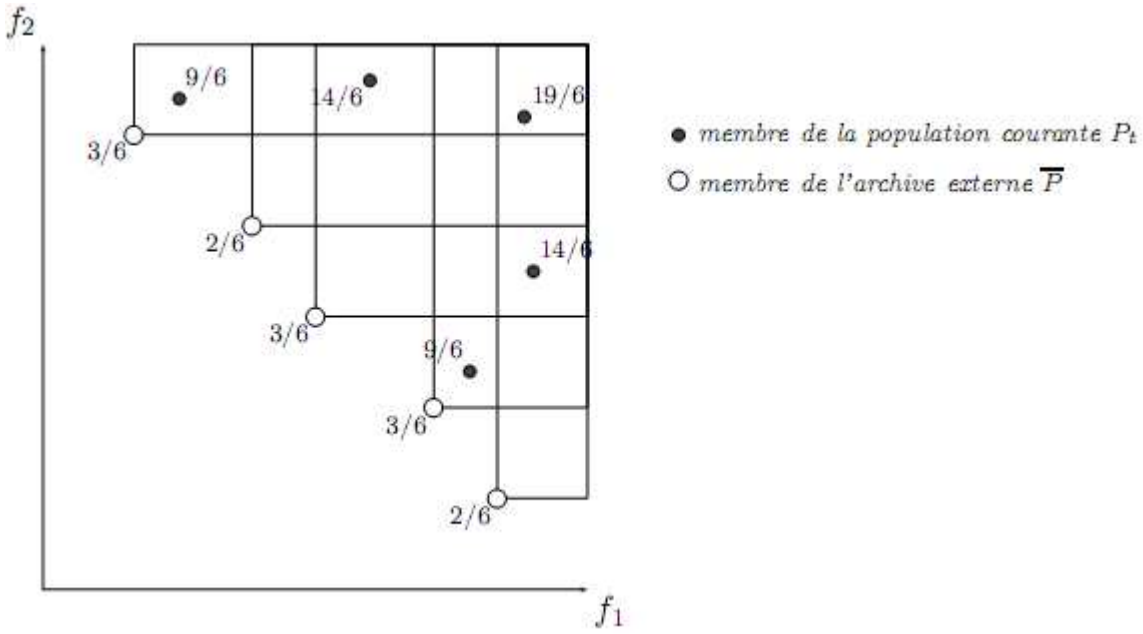


Fig. 3.8 Calcul des valeurs d'adaptation en dimension 2.

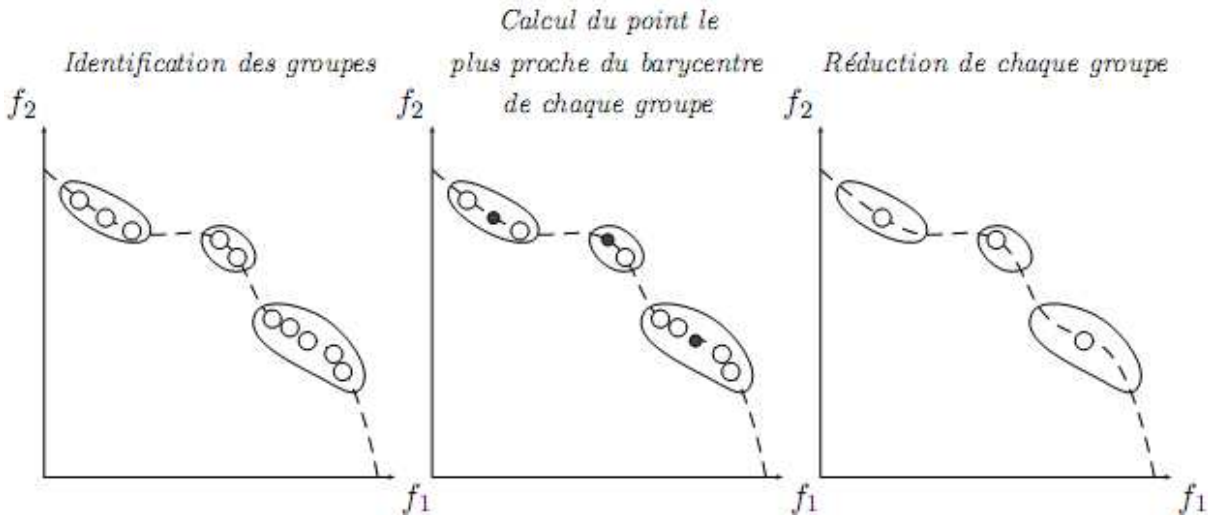


Fig. 3.9 Illustration du clustering en dimension 2.

3.5. Mesure de performance

L'existence de plusieurs solutions optimales (formant la frontière Pareto) et l'absence d'ordre total entre les solutions rendent la mesure de qualité d'un front difficile. En effet, si la notion de dominance au sens de Pareto peut être utilisée pour comparer deux solutions, bien que ces deux solutions puissent être incomparables, la comparaison d'un ensemble de solutions est encore plus délicate. Pourtant lorsque l'on cherche à évaluer un algorithme en terme de

qualité des solutions obtenues, il est nécessaire soit de pouvoir qualifier un front soit de le comparer de façon quantitative avec les fronts produits par d'autres algorithmes. Malheureusement cette tache est délicate puisque la notion de qualité d'un front est elle-même multi-objectif. En effet, un front intéressant est un front qui montre une intéressante convergence vers le front optimal ET une bonne distribution des solutions. Certains fronts peuvent alors être très bons au regard de l'un des objectifs sans être intéressants pour le second.

De nombreux indicateurs de performances ont été proposés dans la littérature. Des articles de synthèse ont été présentés ([83,84]) et en particulier, une étude récente analyse ces indicateurs et expose leurs limitations [85].Voici certaines de ces mesures classées en fonction de leur objectif:mesurer la qualité d'un front isolé, comparer deux fronts...

3.5.1 Indicateurs de qualité s'appliquant à un seul front

L'objectif de ces mesures est de fournir une valeur numérique donnant des indications sur la diversité et/ou la distribution des solutions composant le front. Ces mesures sont très utilisées dans la littérature car elles permettent de qualifier un front indépendamment d'autres fronts. Pourtant elles sont souvent à utiliser avec précaution car ne permettent pas, en général, d'utiliser les valeurs obtenues pour comparer différents fronts. En voici quelques-unes.

Onvg-Overall Non-Dominated Vector Generation

Cette mesure comptabilise le nombre de solutions non dominées générées par l'algorithme [84]. Cette mesure indépendante, facile à calculer, doit être manipulée avec précaution si elle est utilisée pour comparer des fronts.

Schott's Spacing Metric

Cette métrique, basée sur un calcul de distance entre les solutions, a pour objectif de mesurer la distribution des solutions le long du front [86]. Utilisée avec d'autres mesures, elle donne une indication intéressante.

Entropie

L'entropie utilise la notion de niche pour évaluer la distribution des solutions sur le front [55].Plus proche de1est la valeur obtenue, meilleure est la distribution.

3.5.2 Mesures utilisant une référence

Ce type de mesures utilise une référence, qui peut être un point ou l'ensemble Pareto optimal (lorsqu'on a la chance de le connaître), pour évaluer la qualité d'un front. Sans vouloir les citer toutes, en voici des exemples.

Métrique S

Proposée par Zitzler[87],cette mesure calcule l'hypervolume de la région multidimensionnelle comprise entre le front et un point de référence. L'inconvénient de cette métrique est que le résultat dépend du point de référence choisi. Ainsi, la difficulté réside dans le choix de ce point qui doit en particulier être dominé par toutes les solutions des fronts.

Ratio d'erreur

Ce ratio compare le front à qualifier avec le front optimal [84].Il dénombre les solutions n'appartenant pas au front optimal. Plus le ratio est faible, meilleur est le front.

Distance par rapport au front optimal

Plusieurs auteurs ont proposé de mesurer la distance entre le front à étudier et le front optimal [88, 86, 84].En fonction des auteurs, ils préconisent d'utiliser la distance minimale, maximale, moyenne...

3.5.3 Mesures comparant deux fronts Pareto

La comparaison de deux fronts permet de comparer deux méthodes différentes. Lorsque le front optimal est connu cela permet également d'avoir une performance absolue de la méthode sous étude.

Mesure de contribution

La mesure de contribution entre deux fronts (Cont(F1,F2)) permet d'évaluer la proportion de solutions Pareto apportée par chacun des fronts [85].Lorsqu'un front est totalement dominé sa contribution est nulle et Cont(F1,F2)+ Cont(F2,F1)=1. Ainsi, une contribution supérieure à 0,5 indique une amélioration du front.

Métrique C

Cette mesure $C(X', X'')$ indique le ratio de solutions du front X'' faiblement dominées par les solutions du front X' [86]. Lorsque $C(X', X'') = 1$, le front X'' est totalement dominé par X'. Le calcul de ce ratio s'effectue grâce à la formule suivante :

Définition La couverture de deux ensembles

Etant donnés deux ensembles de solutions X' et X''

$$C(X', X'') = \frac{\left|\{a'' \in X'' : \exists a' \in X', a' \succ= a''\}\right|}{\left|X''\right|}$$

La valeur de $C(X', X'') = 1$, signifie que tous les points dans X'' sont dominés ou égaux aux points dans X', tandis que $C(X', X'') = 0$, signifie qu'aucun point dans X'' n'est dominé par un point de X'. Il est à noter qu'il est nécessaire de considérer les deux valeurs $C(X', X'')$ et $C(X'', X')$ car $C(X', X'') \neq 1 - C(X'', X')$.

3.5.4 Conclusion sur l'analyse de performances

Comme nous l'avons vu un ensemble de mesures ont été proposées pour analyser les performances des algorithmes multi-objectifs. Pour une bonne analyse, différentes mesures doivent être utilisées afin de pouvoir analyser à la fois convergence et diversité. L'analyse de performances en multi-objectif est en elle seule un domaine d'étude encore très ouvert puisqu'il n'existe pas de mesure universellement utilisée. Ceci est expliqué par la nature multi-objectif du problème puisque l'on cherche à obtenir des fronts qui approximent le mieux le front optimal et ce suivant différents critères tels que la qualité et la diversification.

3.6. Conclusion

Nous avons défini dans ce chapitre les problèmes d'optimisation multi-objectifs et les éléments de base relatifs. Nous avons présenté plusieurs approches heuristiques qui ont été proposées pour résoudre ces problèmes.

Chapitre 4
Metaheuristique D'optimisation Par Colonies De Fourmis

La place des fourmis dans l'étude des sociétés animales est centrale car elles ont développé des formes très avancées de socialité allant jusqu'à partager leur activité de reproduction en confiant la transmission de leurs gènes à quelques individus de la colonie.

4.1 Optimisation par Colonies de fourmis

Le principe de l'optimisation par colonies de fourmis est apparu au début des années 90. Il est du aux chercheurs M. Dorigo, V. Maniezzo et A. Colorni qui expliquent leur théorie dans un article fondateur [87]. Article dans lequel ils proposent une nouvelle approche pour l'optimisation stochastique combinatoire et mettent en avant la rapidité de leur nouvelle méthode à trouver des solutions acceptables tout en évitant des convergences prématurées. Ils qualifient leur méthode de versatile (elle peut s'appliquer à des versions similaires d'un même problème), robuste et bien sur basée sur une population d'individus [88,89,90,91,92,93,94,95].

Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

L'optimisation par colonies de fourmis s'inspire du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture. Les fourmis en se déplaçant déposent des phéromones, substances olfactives et volatiles. Chaque fourmi se dirige en tenant compte des phéromones qui sont déposées par les autres membres de la colonie. Les fourmis choisissent leur chemin de manière probabiliste. Comme les phéromones s'évaporent progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue continuellement.

4.2. Les fourmis réelles

Les fourmis constituent à l'heure actuelle un support majeur pour les théories développées en écologie comportementale et en sociobiologie. On peut citer plusieurs raisons à cet engouement :

* L'influence des fourmis sur leur environnement naturel est extrêmement importante. Il a par exemple été montré qu'elles déplacent plus de terre en foret tropicale que les vers de terre, ou encore que le poids total des fourmis sur terre est du même ordre de grandeur que le poids des humains. De plus, la domination des fourmis est une preuve de leur adaptation `a des environnements très variés : Dans la foret vierge amazonienne du Brésil, le poids sec de l'ensemble des fourmis est environ quatre fois supérieur `a celui de tous les vertébrés terrestres (mammifères, oiseaux, reptiles et amphibiens) réunis [96]. On trouve ainsi des fourmis dans tous les écosystèmes terrestres situés entre les deux cercles polaires. La connaissance de leur mode de vie est donc primordiale à la compréhension des espèces animales et végétales qui les côtoient;

* L'étude des fourmis se fait assez facilement en laboratoire car elles s'adaptent sans trop de difficultés à des environnements différents de leur habitat d'origine ;

* Les fourmis possèdent une gamme de comportements très variés, collectifs ou individuels.

4.2.1. Les insectes sociaux

La place des fourmis dans l'étude des sociétés animales est centrale car elles ont développé des formes très avancées de socialité allant jusqu'à partager leur activité de reproduction en confiant la transmission de leurs gènes `a quelques individus de la colonie (les reines et les males) [97,98].

Le nombre d'espèces sociales (environ 13 500 connues [96]) est assez réduit par rapport au nombre d'espèces d'insectes répertoriées, soit environ 750 000, alors que les insectes sociaux représentent la moitié de la biomasse des insectes. La grande diversité des fourmis (environ 10 000 espèces connues [96]) propose une large variété de morphologies et de comportements. L'étude des fourmis, la myrmécologie, est donc un vaste et passionnant champ d'investigation.

Une fourmilière peut aussi être assimilée à un super organisme s'apparentant à un organisme vivant composé de cellules. Chaque cellule remplit un rôle précis tout comme les fourmis accomplissent certaines taches pour la survie et le développement du nid. Les fourmis sexuées ont par exemple le même rôle que les cellules sexuelles. Les parallèles entre ces deux types de systèmes sont étonnants et l'étude de leur développement, la morphogenèse pour un organisme et la sociogenèse pour une société d'insectes, permet de faire certains rapprochements. L'étude de la sociogenèse a l'avantage d'être plus facile à réaliser puisque l'on peut retirer et injecter des constituants sans mettre en péril le développement du super organisme.

4.2.2. L’intelligence collective des fourmis

De par la grande diversité des écosystèmes colonisés (des forets vierges aux déserts), les fourmis offrent une grande diversité de comportements et de morphologies [98]. L'étude précise de leur comportement (l'éthologie) est souvent limitée aux espèces les moins populeuses pour des raisons pratiques évidentes d'étude en laboratoire. Cette diversité exubérante est une mine d'inspiration fascinante pour les systèmes informatiques. C'est ainsi que les capacités des fourmis en matière de coopération, de communication, de compétition et d'apprentissage, entre autres, peuvent être mises à profit pour la conception de robots ou d'algorithmes de résolution de problèmes.

4.2.2.1. La communication

Les insectes sociaux en général, et les fourmis en particulier, ont développé des mécanismes de communication très élaborés [99] pour les insectes sociaux pour les animaux en général). Il a été défini douze types de réponse mettant en oeuvre une forme de communication [100] :

1. L'alarme ;

2. L'attraction simple ;

3. Le recrutement (pour une source de nourriture ou un site de nidification);

4. L'entretien et la mue ;

5. La trophallaxie (échange de liquides) ;

6. L'échange d'aliments solides ;

7. Les effets de groupe (augmentation ou inhibition d'une activité) ;

8. La reconnaissance des apparentés ou de caste ;

9. La détermination de caste ;

10. La compétition pour la reproduction ;

11. Le marquage du territoire et du nid ;

12. La reproduction (différenciation du sexe, de l'espèce, de la colonie...).

La communication chimique est de loin la plus présente chez les fourmis. Les phéromones (mélange d'hydrocarbures) sont `a la base de la communication de nombreuses espèces. La chémoréception présente les avantages suivants :

La diversité des molécules pouvant intervenir permet de fournir des informations qualitatives;

La stabilité du signal pour une molécule peu volatile permet d'assurer une certaine permanence.

Par contre, les principaux inconvénients de la communication chimique sont les suivants :

Elle n'offre que peu d'informations sur la direction ;

Sa propagation est relativement lente et elle est peu adaptée pour la transmission de messages urgents ou pour l'intégration de deux stimulations successives sous une forme temporelle.

Les ouvrières sont par exemple capables de déposer des traces chimiques sur le trajet qu'elles empruntent pour ramener de la nourriture. Au delà du fait que ce marquage leur permet de retrouver leur chemin jusqu'à la fourmilière pour ce qui est du retour et jusqu'à la source de nourriture pour ce qui est d'exploiter une source abondante, cela leur permet de transmettre à leur congénères l'emplacement de l'aubaine.

La communication chimique est aussi mise à l'oeuvre pour déclencher des alarmes quand le nid est attaqué et ainsi mobiliser un grand nombre d'individus pour défendre la fourmilière.

Ces deux mécanismes font partie des comportements de recrutement. De plus, plusieurs phéromones peuvent être utilisées et avec des concentrations différentes, constituant ainsi une sorte de langage chimique. Les principales manifestations du recrutement sont la recherche de nourriture, la construction du nid, la défense de la colonie et la migration vers de nouveaux sites de nidification.

Bien que peu répandue, certaines espèces ont développé une forme de communication acoustique soit en utilisant un grattoir ou en utilisant les vibrations du sol. Les mouvements peuvent aussi servir de canal de communication : certaines fourmis tisserandes se livrent `a une sorte de danse pour recruter des ouvrières. On trouve aussi des ouvrières qui transportent d'autres ouvrières pour leur indiquer le nouvel emplacement du nid [96]. La communication tactile entre aussi en jeu dans de nombreux rituels d'invitation et de recrutement. Enfin la communication visuelle est assez difficile à mettre en évidence mais certaines espèces semblent utiliser ce canal pour déclencher des mouvements collectifs notamment lors de l'attaque de proies.

La communication entre les individus peut se faire directement ou indirectement.

L'utilisation des phéromones est majoritairement une forme indirecte puisque l'échange d'information se fait grâce au support du sol. Quand deux individus interagissent indirectement en modifiant l'environnement on parle de stigmergie. Ce terme a été introduit par Grassé à propos des mécanismes collectifs de construction du nid chez les termites.

Les différentes applications informatiques qui découlent des capacités de communication des fourmis se retrouvent par exemple en optimisation combinatoire ou la coopération stigmergétique s'applique parfaitement à la recherche du plus court chemin dans un graphe. Ces applications seront détaillées dans le chapitre suivant.

4.2.2.2. La division du travail

Une des caractéristiques particulièrement intéressante est la capacité des sociétés d'insectes à se partager le travail. Les taches que doivent accomplir les ouvrières sont en effet multiples :

* La recherche de nourriture ;

* La défense du nid ;

* L'entretien et la construction du nid ;

* L'entretien des larves et leur approvisionnement en nourriture.

Toutes ces activités, dont l'importance est variable dans le temps et l'espace, doivent être assurées simultanément pour la survie et le développement de la colonie. C'est essentiellement la plasticité de l'organisation déployée par les fourmis qui nous intéresse. Il a été mis en évidence que certains groupes d'individus se spécialisent dynamiquement pour une tache particulière. Cette dynamique peut être mise en oeuvre pour un individu particulier : sa tache de prédilection varie dans le temps, dans ce cas toutes les ouvrières sont potentiellement capables d'accomplir n'importe quelle tache. On trouve aussi des spécialisations morphologiques, avec par exemple des variations de taille de un `a dix `a l'intérieur de la même espèce. Dans ce cas la dynamique est assurée par un contrôle des naissances sur chaque type de morphologie.

4.2.2.3. La construction du nid

L'architecture des nids construits par les fourmis est un exemple frappant de structure complexe. L'intérêt pour des modèles pouvant expliquer l'apparition de telles structures provient encore une fois de l'organisation distribuée qui est sous-jacente. Il n'y a pas, a priori, de contrôle centralisé, de coordination de niveau supérieur à l'individu. La structure émerge des interactions inter- individuelles et avec l'environnement. La communication indirecte entre les individus est la encore mise à profit.

Les fourmis tisserandes sont par exemple capables d'unir leurs efforts pour rapprocher des feuilles en formant de véritables ponts puis d'unir les bords des feuilles en utilisant la soie produite par leurs larves. La construction du nid chez les termites a été étudiée par Deneubourg. L'apparition des piliers dans une termitière pouvant être expliquée par l'amplification de multiples fluctuations chaotiques : la structure, modèle d'équilibre des forces par sa stabilité, naît de l'amplification de multiples déséquilibres. La construction des nids de guêpes est aussi un modèle d'action collective ou les agents répondent plus particulièrement à des stimuli issus de certaines configurations dans la structure.

4.2.2.4. La quête de nourriture

La recherche de la nourriture (le fourragement) est une activité souvent plus dispersée spatialement que la construction du nid et qui peut aussi être mise en oeuvre de facon tr`es différente suivant les espèces de fourmis. Les stratégies de recherche de nourriture sont en effet extrêmement diversifiées. Par exemple `a cause des différences de régime alimentaire : certaines espèces peuvent être spécialisées sur un unique type d'aliment. On peut aussi trouver des mécanismes très élaborés comme la culture de champignon ou l'élevage de pucerons. La recherche de nourriture est une activité risquée (les fourrageuses de Cataglyphis bicolor ont par exemple une espérance de vie de 6.1 jours [96])) mais souvent efficace (la quantité de nourriture ramenée au nid au cours d'une vie représente de 15 `a 20 fois le poids d'un individu). La communication peut avoir un impact important, en particulier pour les mécanismes de recrutement dont le principal intérêt collectif est de rassembler les ouvrières sur les sources de nourriture rentables. D'un point de vue plus général, la communication mise en oeuvre pour la recherche de nourriture peut être considérée comme une forme de mémoire collective quand elle s'appuie sur la modification de l'environnement telle que l'utilisation des phéromones.

4.2.2.5. Capacités individuelles

Les capacités individuelles des fourmis peuvent servir de modèle à des systèmes artificiels tant leur adaptation à leur environnement peut être efficace. Nous citons par exemple les points suivants :

* Individuellement, une fourmi possède certaines capacités d'apprentissage, et notamment quand elle se déplace autour du nid. Les expériences de Schatz et ses collègues montrent que les fourmis du genre Cataglyphis sont capables d'apprendre visuellement des routes familières pour se déplacer entre un site alimentaire et leur nid;

* Du point de vue physique, certaines espèces ont des capacités étonnantes comme les fourmis Gigantiops destructor capables de faire des bonds impressionnants et dotées de capacités visuelles inhabituelles ce qui les a rendues difficiles à observer.

La plupart des caractéristiques qui intéressent l'informatique sont cependant collectives. Les caractéristiques individuelles ne sont évidemment pas une particularité des fourmis mais de tous les organismes vivants ayant un souci de survie.

4.2.3. Les comportements collectifs des insectes

4.2.3.1. L'auto organisation chez les insectes sociaux

Les théories rassemblées sous le terme d'auto organisation ont originellement été développées en physique ou en chimie pour décrire l'émergence de phénomènes macroscopiques à partir d'interactions et de processus définis au niveau microscopique. L'auto organisation se prête bien à l'étude des insectes sociaux qui montrent des comportements collectifs complexes issus de comportements individuels simples. On peut regrouper les processus d'auto organisation chez les insectes sociaux en quatre groupes tant leur diversité est importante :

* Le recrutement et l'exploitation collective de sources de nourriture : le fourragement met à jour des stratégies qui permettent aux insectes une grande adaptation à leur milieu ;

* La division du travail et l'organisation des rôles sociaux : à l'intérieur d'une même société, on peut observer différentes castes spécialisées dans un certain nombre de taches (élevage du couvain, recherche de nourriture, construction du nid, ...) ;

* L'organisation de l'environnement : la construction du nid est un symbole de l'organisation distribuée des insectes. Le nid est construit sans que les insectes soient dirigés, ils répondent à un certain nombre de stimuli provenant de leur environnement ;

* La reconnaissance inter-individuelle : chaque fourmi est capable d'identifier ses congénères tout en participant elle même `a l'identité de sa colonie (l'échange d'aliments entre les individus d'une même colonie, la trophallaxie, est un exemple d'acte altruiste permettant en plus d'homogénéiser l'identité de la colonie, l'identité coloniale). Les explications du mécanisme de reconnaissance ne sont pas encore parfaitement établies. Cependant, il s'avère qu'il y ait `a la fois une composante génétique et une composante acquise par apprentissage. Un réseau de neurones a par exemple été utilisé pour reproduire ce mécanisme d'apprentissage puis de différenciation entre les composés chimiques, dans le cas des termites.

Ces processus d'auto-organisation sont à l'origine de ce que l'on dénomme l'intelligence collective. On parle d'intelligence collective quand un groupe social peut résoudre un problème dans un cas ou un agent isolé en serait incapable.

4.2.3.2. Stigmergie

La stigmergie est un des concepts à la base de la création des méta- heuristiques de colonies de fourmis. Elle est précisément définie comme une «forme de communication passant par le biais de modifications de l'environnement», mais on peut rencontrer le terme « interactions sociales indirectes » pour décrire le même phénomène. La spécificité de la stigmergie est que les individus échangent des informations par le biais du travail en cours, de l'état d'avancement de la tache globale à accomplir [101].

4.2.3.3. Contrôle décentralisé

Dans un système auto- organisé, il n'y a pas de prise de décision à un niveau donné, suivie d'ordres et d'actions prédéterminées. En effet, dans un système décentralisé, chaque individu dispose d'une vision locale de son environnement, et ne connaît donc pas le problème dans son ensemble. La littérature des systèmes multi- agents emploie souvent ce terme ou celui « d'intelligence artificielle distribuée », bien que, d'une manière générale, l'étude des systèmes multi agents tende à utiliser des modèles de comportement plus complexes, fonde notamment sur les sciences de la cognition. Les avantages d'un contrôle décentralise sont notamment la robustesse et la flexibilité : systèmes robustes, car capables de continuer à fonctionner en cas de panne d'une de leurs composantes ; flexibles, car efficaces sur des problèmes dynamiques.

4.2.3.4. Hétérarchie dense

L'hétérarchie dense est un concept issu directement de la biologie, utilisé pour décrire l'organisation des insectes sociaux, et plus particulièrement des colonies de fourmis [96]. La définition peut être traduite comme suit :

Une colonie de fourmis est une variante particulière de hiérarchie qui peut avantageusement être appelée une hétérarchie. Cela signifie que les propriétés des niveaux globaux agissent sur les niveaux locaux, mais que l'activité induite dans les unités locales influence en retour les niveaux globaux. L'hétérarchie est dite dense dans le sens ou un tel système forme un réseau hautement connecte, ou chaque individu peut échanger des informations avec n'importe quel autre. Ce concept est en quelque sorte opposé à celui de hiérarchie ou, dans une vision populaire mais erronée, la reine gouvernerait ses sujets en faisant passer des ordres dans une structure verticale, alors que, dans une hétérarchie, la structure est plutôt horizontale Fig 4.1.

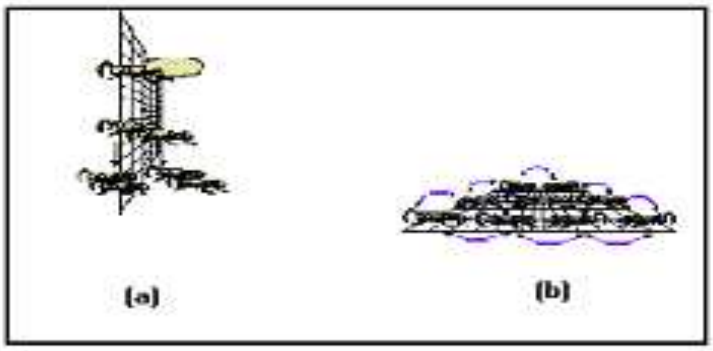


Fig. 4.1: [a] Hiérarchie dense ; [b] Concept opposé

On constate que ce concept recoupe celui de contrôle décentralisé, mais aussi celui de stigmergie, en ce sens que l'hétérarchie décrit la manière dont le flux d'information parcourt le système. Cependant, dans une hétérarchie dense, tout type de communication doit être pris en compte, tant la stigmergie que les échanges directs entre individus.

4.2.3.5. Les pistes de phéromones

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées phéromones. Elles sont attirées par ces substances, qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères Figure 4.2.

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter, sans que les individus aient une vision globale du trajet.

En effet, comme l'illustre la figure 4.2, les fourmis le plus rapidement arrivées au nid, après avoir visite la source de nourriture, sont celles qui empruntent les le chemin le plus court. Ainsi, la quantité de phéromone présente sur le plus court trajet est légèrement plus importante que celle présente sur le chemin le plus long. Or, une piste présentant une plus grande concentration en phéromone est plus attirante pour les fourmis, elle a une probabilité plus grande d'être empruntée. La piste courte va alors être plus renforcée que la longue, et, à terme, sera choisie par la grande majorité des fourmis.

On constate qu'ici le choix s'opère par un mécanisme d'amplification d'une fluctuation initiale. Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'expérience, la colonie choisisse le plus long parcours.

D'autres expériences, avec une autre espèce de fourmis, ont montré que si les fourmis sont capables d'effectuer des demi-tours sur la base d'un trop grand écart par rapport à la direction de la source de nourriture, alors la colonie est plus flexible et le risque d'être piégé sur le chemin long est plus faible.

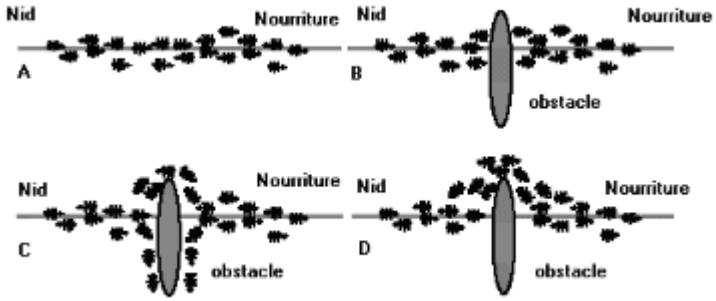


Fig.4.2. Les Fourmis réelles suivent un chemin entre le Nid et la Nourriture. (B) Un obstacle apparaît sur le chemin : Les Fourmis choisissent de tourner soit à gauche soit à droite avec une probabilité égale. (C) La phéromone est déposé plus rapidement sur le chemin le plus court. (D) Toutes les fourmis ont choisi le chemin le plus court.

Il est difficile de connaître avec précision les propriétés physico-chimiques des pistes de phéromone, qui varient en fonction des espèces et d'un grand nombre de paramètres. Cependant, les métas- heuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomène d'évaporation des pistes de phéromone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet « d'heuristiques » leur apportant un peu plus d'informations sur le problème (par exemple une information sur la direction). Il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture) peut être en concurrence avec l'intérêt adaptatif de tels comportements. Si l'on prend en compte l'ensemble des contraintes que subit une colonie de fourmis (prédation, compétition avec d'autres colonies, etc.), un choix rapide et stable peut être meilleur, et un changement de site exploite peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une telle option.

4.3. Les fourmis artificielles

La fourmi artificielle se présente sous la forme d'un ensemble de procédures qui définissent son comportement [102,103]. Celui-ci est très semblable à celui de la fourmi naturelle quand elle recherche de la nourriture. Dans ce cas, une fourmi n'a qu'un rôle assez simple qui consiste à se déplacer du nid jusqu à la source de nourriture et à y revenir. Le code qui définit leur comportement permet aux fourmis artificielles de se déplacer dans l'espace combinatoire formé par les différents éléments qui peuvent être utilisés pour le problème à résoudre. Pour utiliser un vocabulaire informatique, nous dirons qu'elle construit une solution. La mémorisation de ces déplacements donne la forme d'une solution où chaque étape est désignée par I'indice de l'élément et où I'ordre de parcours désigne la position des éléments dans la solution.

1. Dans le cas des fourmis artificielles, il n'y a pas obligatoirement de phase de recrutement.

1. Qui est capable d'agir dans un environnement;

2. Qui peut communiquer directement avec d'autres agents;

3. Qui, est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser) ;

4. Qui possède des ressources propres;

5. Qui, est capable de percevoir (mais de manière limitée) son environnement ;

6. Ne dispose que d'une représentation partielle et éventuellement nulle de cet environnement;

7. Qui, possède des compétences et offre des services;

8. Qui, peut éventuellement se reproduire;

9. Dont le comportement tend à, satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

En se basant sur ces définitions, il est possible de décrire le comportement des fourmis virtuelles en tant qu'agents :

1- Les entités informatiques que nous venons de définir sont dites virtuelles car elles n'ont pas d'existence matérielle au contraire des entités physiques qui interagissent dans le monde concret (Des exemples de ces agents physiques sont un robot, un avion, une voiture) ;

2- Les agents sont capables d'agir : dans le cas des fourmis virtuelles, elles modifient les valeurs de phéromone associées aux différents éléments. Par cette action elles changent leur environnement, ce qui influera sur le choix des autres fourmis à l'itération suivante ;

3- Les agents sont capables de communiquer : les fourmis utilisent comme on I'a vu précédemment la phéromone comme médium de communication indirecte;

4- les agents sont doués d'autonomie : chaque fourmi â pour but de construire une solution pour un problème donné, se contentant pour cela d'appliquer les règles de sélection qui définissent son comportement, la fourmi utilise la phéromone et parfois des valeurs heuristiques;

5- Les agents n'ont qu'une représentation partielle de leur environnement : lors de la construction d'une solution, la fourmi ne connaît à chaque étape que les éléments qu'elle a déjà choisis et les valeurs de phéromone correspondant aux éléments qui pourront l'être.

Comme nous avons pu le voir, les fourmis peuvent résoudre collectivement des problèmes complexes. Un des exemples marquant est celui de la découverte du chemin le plus court dans l'expérience du choix entre les deux chemins. Chacune des fourmis, isolément, ne peut trouver la meilleure solution. C'est grâce au mécanisme d'auto- organisation qui émerge de la communication indirecte (stigmergie) que le plus court chemin peut être découvert. Le modèle de la stigmergie peut être facilement étendu aux agents artificiels en associant aux éléments d'un problème des variables d'état spécifique, qui serviront de support à la communication indirecte entre fourmis.

Pour simuler ce mécanisme, il faut intégrer un processus de renforcement positif. Dans le cas des fourmis, plus le chemin est court, plus les fourmis le parcourront vite et plus la quantité de phéromone associée sera élevée. Afin de pouvoir comparer les solutions, il est nécessaire d'introduire une mesure qui donne l'adéquation de la solution au problème : c'est la fonction

qualité (en anglais fitness). La valeur de cette qualité est utilisée dans le calcul de la mise à jour des valeurs associées aux éléments du problème lors de la phase de renforcement. Pour la recherche chemin le plus court, on peut prendre tout simplement la distance comme mesure de qualité : plus le chemin est court, plus il sera renforcé.

Une des propriétés de la phéromone est son caractère volatil. Dans le modèle virtuel, un mécanisme similaire sera utilisé afin d'éviter une convergence prématurée (stagnation) due à la découverte d'un optimum local.

Entre ces deux modèles, nous passons d'un univers continu à un univers discret. Ainsi, les fourmis virtuelles sautent d'un élément à un autre, tandis que les fourmis naturelles progressent de façon continue sur le chemin. Cette discrétisation impose que l'évaporation ou la modification des valeurs de phéromone ne puisse se faire en continu. Cet ajustement de valeurs n'est souvent réalisable qu'après la construction de la solution complète.

4.4. Optimisation par colonies de fourmis

Le problème du voyageur de commerce (Travelling Salesman Problem, **TSP**) a fait l'objet de la première implémentation d'un algorithme de colonies de fourmis : le [Ant Syste (AS)] [89].

Le problème du voyageur de commerce consiste à trouver le trajet le plus court (désigne par «tournée» ou plus loin par «tour») reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement défini comme un graphe complètement connecte(N, A), ou les villes sont les noeuds *N* et les trajets entre ces villes, les arêtes *A*.

4.4.1 Algorithme de base

Dans l'algorithme AS, à chaque itération$t(1 \quad t \quad t_{\max})$, chaque fourmi $K(K = 1,.....,m)$parcourt le graphe et construit un trajet complet de $n = |N|$étapes (on note $|N|$ le cardinal de l'ensemble N). Pour chaque fourmi, le trajet entre une ville i et une ville j dépend de :

La liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i : J_i^K ;

L'inverse de la distance entre les villes : $\eta_{ij} = \frac{1}{d_{ij}}$, appelée visibilité. Cette information «statique» est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines ;

La quantité de phéromone déposée sur l'arête reliant les deux villes, appelée l'intensité de la piste. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage.

La règle de déplacement (appelée «règle aléatoire de transition proportionnelle» par les auteurs) est la suivante :

$$P_{ij}^{K}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha} \ (\eta_{ij})^{\beta}}{\sum_{l \ J_i^K} (\tau_{il}(t))^{\alpha} \ (\eta_{il})^{\beta}} \text{.....si...} j \quad J_i^K \\ \mathbf{0} \end{cases}$$

Ou αet βsont deux paramètres contrôlant l'importance relative de l'intensité de la piste,

$\tau_{ij}(t)$, et de la visibilitéη_{ij}. Avec$\alpha = 0$, seule la visibilité de la ville est prise en compte; la ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de phéromone jouent. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres, jouant sur les comportements de diversification et d'intensification est nécessaire.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromone

$\Delta\tau_{ij}^{K}(t)$sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{K}(t) = \begin{cases} \dfrac{Q}{L^K(t)} \text{.....si.}(i,j) \quad T^K(t) \\ \mathbf{0}\text{.....si.}(i,j) \quad T^K(t) \end{cases}$$

Où

$T^K(t)$: est le trajet effectué par la fourmi kà l'itération$t$

$L^K(t)$: la longueur de la tournée et Qun paramètre fixé.

L'algorithme ne serait pas complet sans le processus d'évaporation des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous optimales, il est nécessaire de permettre au système «d'oublier» les mauvaises solutions. On contrebalance donc l'additivité des pistes par une décroissance constante des valeurs des arêtes à chaque itération. La règle de mise à jour des pistes est donc :

$$\tau_{ij}(t+1) = (1-\rho)\ \tau_{ij}(t) + \Delta\tau_{ij}(t)$$

Ou $\Delta\tau_{ij}(t) = \sum_{K=1}^{m} \Delta\tau_{ij}^{K}(t)$ et m est le nombre de fourmis. La quantité initiale de phéromone sur les arêtes est une distribution uniforme d'une petite quantité $\tau_0 \quad 0$.

La figure 4.3 présente un exemple simplifié de problème du voyageur de commerce.

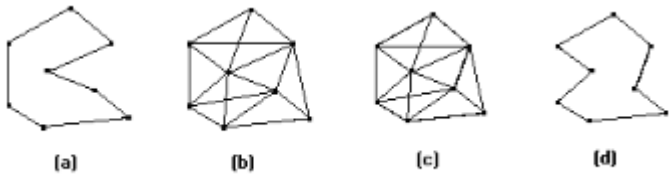


Fig. 4.3 : Le problème du voyageur du commerce, les points représente les villes et l'épaisseur des arêtes la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi. (b) au début du calcul, tous les chemins sont explorés. (c) le chemin le plus court est plus renforcé que les autres. (d) l'évaporation permet d'éliminer les moins bonne solutions.

4.4.2. Choix des paramètres

Pour l'algorithme *AS*, les auteurs préconisent que, bien que la valeur de *Q* ait peu d'influence sur le résultat final, cette valeur soit du même ordre de grandeur qu'une estimation de la longueur du meilleur trajet trouvé. D'autre part, la ville de départ de chaque fourmi est typiquement choisie par un tirage aléatoire uniforme, aucune influence significative du placement de départ n'ayant pu être démontrée.

En ce qui concerne l'algorithme ACS, les auteurs conseillent d'utiliser $\tau_0 = (n\ L_{nn})^{-1}$, ou $n$est le nombre de villes et L_{nn}la longueur d'un tour trouvé par la méthode du plus proche voisin.

Le nombre de fourmis *m* est un paramètre important ; les auteurs suggèrent d'utiliser autant de fourmis que de villes (i.e. *m* = *n*) pour de bonnes performances sur le problème du voyageur de commerce. Il est possible de n'utiliser qu'une seule fourmi, mais l'effet d'amplification des longueurs différentes est alors perdu, de même que le parallélisme naturel de l'algorithme, ce qui peut s'avérer néfaste pour certains problèmes. En règle générale, les algorithmes de colonies de fourmis semblent assez peu sensibles à un réglage fin du nombre de fourmis.

4.4.3. Formalisation d'un algorithme de colonie de fourmis

4.4.3.1. Représentation du problème

Le problème est représenté par un jeu de solutions, une fonction objective assignant une valeur à chaque solution et un jeu de contraintes. L'objectif est de trouver l'optimum global de la fonction objectif satisfaisant les contraintes. Les différents états du problème sont caractérisés comme une séquence de composants. On peut noter que, dans certains cas, un coût peut être associe à des états autres que des solutions[104,105,106].

Dans cette représentation, les fourmis construisent des solutions en se déplaçant sur un graphe *G = (C; L),* ou les noeuds sont les composants de *C* et où l'ensemble *L* connecte les composants de *C*. Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

4.4.3.2. Comportement des fourmis

Les fourmis artificielles peuvent être caractérisées comme une procédure de construction stochastique construisant des solutions sur le graphe *G = (C; L)*. En général, les fourmis tentent d'élaborer des solutions faisables mais, si nécessaire, elles peuvent produire des solutions infaisables. Les composants et les connexions peuvent être associés à des pistes de phéromone τ(mettant en place une mémoire adaptative décrivant l'état du système) et à une valeur heuristique η(représentant une information a priori sur le problème, ou venant d'une source autre que celle des fourmis ; c'est bien souvent le coût de l'état en cours). Les pistes de

phéromone et la valeur de l'heuristique peuvent être associées soit aux composants, soit aux connexions figure 4.4.

Chaque fourmi dispose d'une mémoire utilisée pour stocker le trajet effectué, d'un état initial et de conditions d'arrêt. Les fourmis se déplacent d'après une règle de décision probabiliste fonction des pistes de phéromone locales, de l'état de la fourmi et des contraintes du problème. Lors de l'ajout d'un composant à la solution en cours, les fourmis peuvent mettre à jour la piste associée au composant ou à la connexion correspondante. Une fois la solution construite, elles peuvent mettre à jour la piste de phéromone des composants ou des connexions utilisées. Enfin, une fourmi dispose au minimum de la capacité de construire une solution du problème.

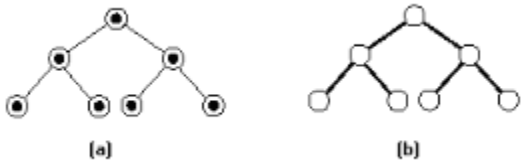


Fig. 4.4 : Dans un algorithme de colonie de fourmis, les piste de phéromone peuvent être associées aux composants (a) ou au connections (b) du graphe représentant le problème à résoudre.

4.4.3.3. Organisation de la méta- heuristique

En plus des règles régissant le comportement des fourmis, un autre processus majeur a cours : l'évaporation des pistes de phéromone. En effet, à chaque itération, la valeur des pistes de phéromone est diminuée. Le but de cette diminution est d'éviter une convergence trop rapide et le piégeage de l'algorithme dans des minimums locaux, par une forme d'oubli favorisant l'exploration de nouvelles régions.

Selon les auteurs du formalisme ACO, il est possible d'implémenter d'autres processus nécessitant un contrôle centralisé (et donc ne pouvant être directement pris en charge par des fourmis), sous la forme de processus annexes. Ce n'est, à notre sens, que peu souhaitable ; en effet, on perd alors la caractéristique décentralisée du système. De plus, l'implémentation de processus annexes entre difficilement dans une formalisation rigoureuse.

4.5. Optimisation par colonies de fourmis pour la résolution de PMO

Récemment plusieurs travaux ont proposé des algorithmes basés sur l'optimisation par colonies de fourmis pour résoudre des PMO, appelés dans la littérature algorithmes MOACO.

4.5.1 Algorithmes MOACO

Nous présentons dans cette section des algorithmes MOACO proposés dans la littérature.

Multiple Objective Ant-Q

L'algorithme "Multiple Objective Ant-Q" (MOAQ) a été proposé par Mariano et Morales dans [107] pour résoudre le problème de distribution de l'eau dans les réseaux d'irrigation. Cet algorithme est basé sur un algorithme d'apprentissage renforcé distribué appelé Ant-Q [108]. L'idée de base de MOAQ est de réaliser un algorithme d'optimisation avec une famille d'agents (fourmis) pour chaque objectif. Chaque famille *k* essaye d'optimiser un objectif en considérant les solutions trouvées pour les autres objectifs et leur fonction correspondante HE^k. De cette manière, tous les agents des différentes familles travaillent dans le même environnement en proposant des actions et une valeur de récompense *r* qui dépend sur comment leurs actions ont participées à trouver des solutions de compromis parmi les autres agents.

MOAQ présente d'autres caractéristiques spécifiques. D'abord, lors de la construction de solution, la $j^{ème}$ fourmi de la $i^{ème}$ famille utilise la solution trouvée par la $j^{ème}$ fourmi de la famille $i - 1$. De plus, lorsqu'une solution trouvée n'est pas faisable, l'algorithme applique une pénalité à ses composants sur les valeurs de Q. Finalement, tout au long du processus, les solutions non-dominées sont enregistrées dans un ensemble externe.

L'algorithme BicriterionAnt

L'algorithme BicriterionAnt a été proposé par Iredi et al. [109] spécialement pour résoudre le problème de tournée de véhicules bi-critère. Pour ce faire, il utilise deux structures de traces de phéromone différentes, τ et τ', une pour chaque critère considéré.

A chaque génération, chacune des *m* fourmis de la colonie génère une solution au problème. Durant l'étape de construction, la fourmi choisit le prochain sommet *j* à visiter relativement à la probabilité suivante :

$$p(j) = \frac{\tau_{ij}^{\lambda\alpha} . \tau'^{(1-\lambda)\alpha}_{ij} . \eta_{ij}^{\lambda\alpha} . \eta'^{(1-\lambda)\alpha}_{ij}}{\sum_{u \in \Omega} \tau_{iu}^{\lambda\alpha} . \tau'^{(1-\lambda)\alpha}_{iu} . \eta_{iu}^{\lambda\alpha} . \eta'^{(1-\lambda)\alpha}_{iu}}$$

Où η_{ij} et η'_{ij} sont les valeurs heuristiques associées à l'arête a_{ij} relativement au premier et second objectif, respectivement, Ω est le voisinage faisable de la fourmi, et λ est calculé pour chaque fourmi $h, f \in 1,...,m$, comme suit :

$$\lambda_h = \frac{h-1}{m-1}$$

Une fois toutes les fourmis ont construit leurs solutions, les traces de phéromone sont évaporées par la règle habituelle. Ensuite, chaque fourmi qui a généré une solution dans le front Pareto au cycle courant est autorisée à mettre à jour les deux structures phéromone τ et τ', en déposant une quantité égale à $1/l$, avec l est le nombre de fourmis qui sont en train de mettre à jour les traces de phéromone. Les solutions non-dominées générées tout au long de l'exécution de l'algorithme sont mises dans un ensemble externe.

L'algorithme BicriterionMC

Iredi et al. ont proposé dans le même travail [109] un autre algorithme MOACO, "BicriterionMC", très similaire au précédent BicriterionAnt. La différence principale est que chaque fourmi met à jour une seule structure de phéromone dans le nouvel algorithme. Les auteurs introduisent une définition générale pour l'algorithme basée sur l'utilisation de *p* structures de traces de phéromone et par la suite mettent *p = 2* pour résoudre des problèmes bi-critères. Pour ce faire, ils considèrent deux méthodes différentes pour la mise à jour des traces de phéromone :

- Méthode 1. *Mise à jour par origine* : une fourmi dépose des traces de phéromone seulement dans sa colonie. Cette méthode force les deux colonies à chercher dans des régions différentes du front Pareto. L'algorithme utilisant cette première méthode est appelé "UnsortBicriterion".
- Méthode 2. *Mise à jour par région* : la séquence des solutions sur le front Pareto est divisée en *p* parties de taille égale. Les fourmis qui ont trouvé des solutions dans la i^{ème} partie dépose de la phéromone pour la colonie $i, i \in [1, p]$. Le but est de guider explicitement les fourmis des colonies de chercher dans des régions différentes

du front Pareto, chacune dans une région. L'algorithme utilisant cette méthode est appelé "BicriterionMC".

Pareto Ant Colony Optimization

"Pareto Ant Colony Optimization" (P-ACO) a été proposé par Doerner et al. Dans [110]. Il a été initialement appliqué au problème de sélection de portefeuille multi-objectif. Il suit le schéma général de l'algorithme ACS, mais la mise à jour globale de phéromone est réalisée en utilisant deux fourmis, la meilleure et la deuxième meilleure solution générée dans le cycle courant pour chaque objectif *k*. Dans P-ACO, plusieurs structures phéromone τ^k sont considérées, une pour chaque objectif. A chaque cycle de l'algorithme, chaque fourmi calcule un ensemble de poids $p=(p_1,...,p_k)$, et l'utilise pour combiner les traces de phéromone et l'information heuristique.

Les solutions non-dominées trouvées tout au long de l'exécution sont là aussi enregistrées dans un ensemble externe.

Multiple Ant Colony System pour le problème de tournées de véhicules avec fenêtres de temps

L'algorithme "Multiple Ant Colony System for Vehicle Routing Problem with Time Windows" (MACS-VRPTW) a été introduit par Gambardella et al. [111].Il utilise, comme P-ACO, le schéma de ACS.

MACS-VRPTW est organisé avec une hiérarchie des colonies de fourmis conçue pour optimiser successivement une fonction multi-objectif : la première colonie, ACS-VEI, minimise le nombre de véhicules alors que la deuxième,ACS-TIME, optimise les solutions faisables trouvées par la première. Chaque colonie utilise une structure phéromone indépendante pour son objectif spécifique et elles collaborent en partageant la meilleure solution globale trouvée ψ^{gb}. Lorsque ACS-VEI est activée, elle essaie de trouver une solution faisable avec un véhicule en moins que ceux utilisés dans ψ^{gb}. Le but de ACS-TIME est d'optimiser le temps total de la tournée des solutions qui utilisent le même nombre de véhicules utilisé dans ψ^{gb}. A chaque fois une meilleure solution est trouvée l'algorithme tue les deux colonies, et le processus est réitéré : deux nouvelles colonies sont activées travaillant avec la nouvelle solution.

Multiple Ant Colony System

Multiple Ant Colony System (MACS) [112] est une variante de MACS-VRPTW décrit dans la section précédente. Il utilise aussi ACS, mais contrairement à son prédécesseur, MACS utilise une seule structure phéromone, τ et plusieurs informations heuristiques, η^k, initialement deux, η^0 et η^1. De cette manière, une fourmi se déplace d'un sommet *i* à un sommet *j* en appliquant la règle suivante :

$$J=\begin{cases} \arg\max_{j\in\Omega}(\eta_{ij}.[\eta_{ij}^0]^{\lambda\beta}.[\eta_{ij}^1]^{(1-\lambda)\beta}) & \text{si} \quad q \le q0 \ ; \\ \hat{i} & \text{sinon} \end{cases}$$

où λ est calculée pour chaque fourmi *h* comme $\lambda = h/m$ avec *m* est le nombre de fourmis, et $\hat{i}$ est le sommet choisi relativement à la probabilité suivante :

$$p(j)=\frac{\tau_{ij}.[\eta_{ij}^0]^{\lambda\beta}.[\eta_{ij}^1]^{(1-\lambda)\beta}}{\sum_{u\in\Omega}\tau_{iu}.[\eta_{iu}^0]^{\lambda\beta}.[\eta_{iu}^1]^{(1-\lambda)\beta}}$$

A chaque fois une fourmi traverse une arête a_{ij}, elle réalise une mise à jour de phéromone locale comme suit :

$\tau_{ij} = (1-\rho).\tau_{ij} + \rho.\tau_0$

Initialement, τ_0 est calculé d'un ensemble de solutions heuristiques en prenant leurs moyennes dans chaque fonction objectif, f_0 et f_1, et en appliquant l'expression suivante :

$$\tau_0 = \frac{1}{\hat{f}_0.\hat{f}1}$$

Cependant, la valeur de τ_0 ne reste pas fixe, comme habituellement dans ACS, mais elle est mise à jour durant l'exécution de l'algorithme, avec la formule précédente avec les nouvelles valeurs des fonctions objectifs des solutions non dominées trouvées.

La mise à jour globale est réalisée avec chaque solution S de l'ensemble Pareto courant en appliquant la règle suivante :

$$\tau_{ij} = (1-\rho).\tau_{ij} + \frac{\rho}{f^0(S).f^1(S)}.$$

COMPETants

Doerner et al. ont introduit l'algorithme COMPETants [113] pour traiter les problèmes de transport bi-objectif. L'algorithme adopte le schéma de "rank-based AS" [108]. Il utilise deux colonies de fourmis, chacune avec sa propre structure de phéromone et sa propre information heuristique. Les deux colonies sont en compétition sur le nombre de fourmis. Le nombre de fourmis dans chaque colonie n'est pas fixe, la colonie qui trouve de meilleures solutions reçoit plus de fourmis dans la prochaine itération. De plus, l'information passe entre les deux colonies, puisque les fourmis observent et utilisent non seulement leur propre trace de phéromone mais aussi la trace étrangère.

La décision des fourmis d'utiliser ou non la trace de phéromone étrangère est basée sur la meilleure solution trouvée dans chaque colonie. Les fourmis qui décident d'utiliser la trace étrangère sont appelées "espions" ou "spies".

Multi-Objective Network ACO

Multi-objective Network ACO (MONACO) [114] a été conçu pour être appliqué à un problème dynamique, l'optimisation du trafic d'un message dans un réseau.

Cet algorithme est différent des autres MOACO présentés puisqu'il traite un problème dynamique, la "policy" du réseau change suivant les étapes de l'algorithme et elle n'attend pas la fin de l'exécution. MOACO suit le schéma de base de AS classique mais utilise plusieurs traces de phéromone et une seule information heuristique. Chaque fourmi, qui est représentée comme un message, utilise pour se déplacer, dans la probabilité de transition, une agrégation des traces de phéromone.

L'objectif de l'algorithme n'est pas de trouver un bon ensemble de solution non-dominées mais de rendre le réseau travaille efficacement.

Crowding Population-based Ant Colony Optimisation

Crowding Population-based Ant Colony Optimisation (CPACO) [115] a été proposé pour résoudre le problème de voyageur de commerce multi-objectif. CPACO étend l'algorithme Population-based Ant Colony Optimisation (PACO) [116] en utilisant, d'abord, un schéma de remplacement de "crowding" (ou surpeuplement)[117]. CPACO utilise une seule structure phéromone avec des structures heuristiques individuelles, alors que PACO utilise des structures phéromone et heuristiques individuelles pour chaque objectif. A chaque itération une nouvelle structure phéromone est calculée comme suit :

- A toutes les solutions (s) dans la population (S) est affecté un rang relativement au rang de dominance défini dans l'algorithme NSGAII [118].
- Tous les éléments dans la structure phéromone sont initialisés à une certaine valeur (τ_{init}).
- Toutes les solutions incrémentent leurs éléments correspondants dans la structure phéromone relativement à l'inverse de leur rang, i.e. $\Delta\tau_{ij}^{S} = 1/(S_{rank})$.

Pour déterminer les poids pour chaque objectif, CPACO affecte à chaque fourmi un ensemble unique de facteurs heuristiques (λ). Ceci permet à chaque fourmi d'exploiter les structures heuristiques avec différentes quantités tout en utilisant une structure phéromone commune.

4.5.2 Taxonomie des algorithmes MOACO

Pour résoudre un problème d'optimisation multi-objectif (PMO) avec ACO, plusieurs points sont à définir pour lesquels différents choix sont possibles. Une taxonomie de plusieurs algorithmes MOACO a été proposée dans [119]. Cette taxonomie classe les algorithmes MOACO par le nombre de structures de phéromone et le nombre d'informations heuristiques. Nous considérons, dans ce travail, que ces algorithmes diffèrent essentiellement dans les points suivants :

Structures de phéromone. Les traces de phéromone représentent le moyen qu'utilise une fourmi pour attirer les autres fourmis de sa colonie vers les aires correspondantes. Pour le cas uni-objectif, ces traces reflètent la désirabilité de visiter une zone de l'espace de recherche suivant la fonction à optimiser. En traitant un PMO, où on a plusieurs objectifs à optimiser simultanément, deux choix s'offrent :

1. On peut utiliser une seule structure de phéromone comme proposé dans [107]. Dans ce cas, les traces de phéromone déposées par une fourmi sont définies relativement à une agrégation des objectifs à optimiser.
2. Une deuxième possibilité consiste à considérer plusieurs structures de phéromone comme proposé dans [109]. Dans ce cas, on peut utiliser plusieurs colonies de fourmis, chacune utilise sa propre phéromone.

Définition du facteur phéromone. A chaque étape de la construction de solution d'une fourmi, un candidat est choisi relativement à une probabilité de transition qui dépend de deux

facteurs : un facteur phéromone et un facteur heuristique. La définition du facteur phéromone dépend de la définition de la structure de phéromone, comme discuté dans le premier point.

1. Lorsqu'une seule structure phéromone est utilisée, le facteur phéromone est défini relativement à cette structure.
2. Lorsque plusieurs structures phéromone sont utilisées, on peut soit utiliser pour chaque objectif à optimiser la structure phéromone correspondante comme proposé dans [113], soit utiliser toutes les structures phéromone. Dans ce dernier cas, généralement une agrégation des structures phéromone est utilisée : une somme pondérée comme proposée dans [110] ou un produit pondéré comme dans [109].

Définition du facteur heuristique. Ce facteur heuristique doit donner une idée sur l'apport du candidat en question à la solution en cours de construction. Deux stratégies différentes ont été considérées :

1. une première stratégie est de considérer une agrégation des différents objectifs dans une seule information heuristique et d'utiliser cette information comme facteur heuristique, comme proposé dans [110],
2. une deuxième stratégie consiste à associer à chaque objectif une structure heuristique. Dans ce cas, comme dans la définition du facteur phéromone, on peut soit utiliser séparément chaque fonction heuristique pour l'objectif correspondant, comme proposé dans [113], soit agréger les différentes structures heuristiques comme dans [107].

Solutions à récompenser. Lors de la mise à jour de phéromone, on doit décider sur quelles solutions construites déposer la phéromone.

1. Une première possibilité, appelée l'approche Pareto, consiste à récompenser les solutions non-dominées de l'ensemble Pareto du cycle courant. Pour cette dernière possibilité on peut encore décider si on va récompenser toutes les solutions de l'ensemble Pareto comme proposé dans [112] ou bien seulement les nouvelles solutions non-dominées qui entrent dans l'ensemble au cycle courant comme dans [109].
2. Une deuxième possibilité, appelée l'approche non Pareto, est de récompenser chacune des solutions qui trouve la meilleure valeur pour chaque critère du cycle courant comme proposé dans [110].

4.7. Discussion

Nous avons présenté dans cette section plusieurs algorithmes MOACO proposés dans la littérature. Ces algorithmes proposent différentes stratégies pour la résolution des problèmes multi-objectifs.

Chapitre 5
Etude de différentes stratégies phéromonales : Cas du problème du sac à dos multidimensionnel

5.1 Introduction

Dans ce chapitre, nous comparons trois stratégies différentes de dépôt et exploitation des traces de phéromone. Nous utilisons comme application un problème d'optimisation combinatoire uni-objectif connu dans la littérature NP-difficile : le problème du sac à dos multidimensionnel (MKP). Ainsi, nous décrivons dans la section suivante l'algorithme ACO proposé pour le MKP, et nous présentons trois variantes différentes de cet algorithme, correspondant aux trois stratégies phéromonales. Nous discutons ensuite brièvement de l'influence des paramètres de l'algorithme sur l'intensification/diversification de la recherche. Enfin, nous comparons expérimentalement les trois stratégies proposées, ainsi que d'autres approches évolutionnaires, sur un ensemble de problèmes issus d'un benchmark classique. L'intérêt de ce travail dépasse le cadre de la résolution du problème du sac-à-dos multidimensionnel. En effet, de nombreux problèmes d'optimisation combinatoire consistent à sélectionner un sous-ensemble d'objets optimisant une fonction objectif donnée tout en respectant un certain nombre de contraintes, e.g., la recherche de cliques maximum, les problèmes de satisfaction de contraintes. Pour résoudre ces problèmes avec la métaheuristique ACO, il s'agit essentiellement de décider sur quels composants de solutions déposer de la phéromone. Ainsi, notre étude comparative sur le MKP pourrait aider à choisir une stratégie pour la résolution d'autres problèmes de ce type. L'essentiel des travaux réalisés dans ce chapitre est publié dans [120] et [121].

5.2 Algorithme ACO pour le MKP

Dans les algorithmes ACO, les fourmis déposent les traces de phéromone sur les composants des meilleures solutions construites pour attirer les autres fourmis vers les aires

correspondantes de l'espace de recherche. Ainsi pour résoudre un problème avec la métaheuristique ACO, un point clé est de décider sur quels composants des solutions construites les traces de phéromone vont être déposées et comment exploiter ces traces lors de la construction de nouvelles solutions. Une solution d'un MKP est un ensemble d'objets sélectionnés $S = \{o_1 ..., o_k\}$ (on considèrera qu'un objet oi est sélectionné si la variable de décision correspondante x_{oi} a été mise à 1). Etant donnée une telle solution S, trois différentes manières de déposer la phéromone peuvent être considérées :

- Une première possibilité est de déposer les traces de phéromone sur chaque objet sélectionné dans S. Dans ce cas, lors de la construction des solutions suivantes, la probabilité de sélectionner chaque objet de S sera augmentée.
- Une deuxième possibilité est de déposer les traces de phéromone sur chaque couple $\{o_i, o_{i+1}\}$ de deux objets successivement ajoutés dans S. Dans ce cas, l'idée est d'augmenter la probabilité de choisir l'objet $\{o_{i+1}\}$ si le dernier objet sélectionné est $\{o_i\}$.
- Une troisième possibilité est de déposer les traces de phéromone sur toutes les paires $\{o_i, o_j\}$ de deux objets appartenant à S. Dans ce cas, lors de la construction d'une nouvelle solution S_0, la probabilité de choisir l'objet o_i sera augmentée si o_j a déjà été sélectionné dans S_0. Plus précisément, plus S_0 contiendra de sommets appartenant déjà à S, et plus les autres sommets de S auront de chances d'être sélectionnés.

Pour comparer ces trois stratégies phéromonales différentes, nous introduisons maintenant l'algorithme Ant-Knapsack (AK) et nous décrivons trois variantes différentes de cet algorithme : Vertex-AK dans lequel la phéromone est déposée sur les sommets, Path-AK dans lequel la phéromone est déposée sur les arcs du chemin visité, et Edge-AK dans lequel la phéromone est déposée sur les arêtes reliant tous les sommets appartenant à une solution.

5.2.1 L'algorithme Ant-Knapsack

L'algorithme Ant-Knapsack est décrit dans la Figure 5.1. A chaque cycle de cet algorithme, chaque fourmi construit une solution. Lorsque toutes les fourmis ont construit une solution, les traces de phéromone sont mises à jour. L'algorithme s'arrête lorsqu'une fourmi a trouvé une solution optimale (si la valeur optimale est connue), ou lorsqu'un nombre maximal de cycles a été atteint. Notons que cet algorithme est inspiré du MAX −MIN Ant System [122]

dans lequel les traces de phéromone sont limitées à l'intervalle [$\tau_{\min}$, $\tau_{\max}$] et sont initialisées à $\tau_{\max}$.

Pour construire une solution, les fourmis choisissent aléatoirement un objet initial, puis ajoutent itérativement des objets qui sont choisis à partir d'un ensemble Candidats qui contient tous les objets qui peuvent être sélectionnés sans violer de contraintes de ressources.

A chaque étape, l'objet o_i à ajouter à la solution en cours de construction S_k est choisi parmi l'ensemble de sommets Candidats relativement à une probabilité $p_{S_k}(o_i)$. Cette probabilité est définie proportionnellement à un facteur phéromonal $\tau_{S_k}(o_i)$et un facteur heuristique $\eta_{S_k}(o_i)$, ces deux facteurs étant pondérés par deux paramètres α et β qui déterminent leur importance relative. Le facteur phéromonal dépend de la stratégie phéromonale choisie et est décrit plus tard. Le facteur heuristique $\eta_{S_k}(o_i)$est défini de façon similaire à [123] et [124] : soit $d_{S_k}(i) = b_i - \sum_{g \in S_k} r_{ig}$ la quantité restante de la ressource *i* lorsque la fourmi a construit la solution S_k ; nous définissons le ratio

$$h_{S_k}(j) = \sum_{i=1}^{m} \frac{r_{ij}}{d_{S_k}(i)}$$ qui présente la dureté de l'objet *j* par rapport à toutes les contraintes $i \in 1..m$ et relativement à la solution construite S_k , de sorte que plus ce ratio est faible plus l'objet est intéressant.

Nous intégrons alors le profit p_j de l'objet *j* pour obtenir un ratio profit/ressource, et nous définissons le facteur heuristique par $\eta_{s_k}(j) = \frac{p_j}{h_{S_k}(j)}$.

Algorithme Ant-Knapsack :

Initialiser les traces de pheromone a τ_{max}

repeter

pour chaque fourmi k **dans** $1..nbAnts$, construire une solution $\mathcal{S}_k$ comme suit :

Choisir aleatoirement un premier objet $o_1 \in 1..n$

$\mathcal{S}_k \leftarrow \{o_1\}$

$Candidats \leftarrow \{o_i \in 1..n / o_i$peut être selectionne sans violer des contraintes de ressources$\}$

tant que $Candidats \neq \emptyset$ **faire**

Choisir un objet $o_i \in Candidats$ avec la probabilite

$$p_{\mathcal{S}_k}(o_i) = \frac{[\tau_{\mathcal{S}_k}(o_i)]^{\alpha}.[\eta_{\mathcal{S}_k}(o_i)]^{\beta}}{\sum_{o_j \in Candidats}[\tau_{\mathcal{S}_k}(o_j)]^{\alpha}.[\eta_{\mathcal{S}_k}(o_j)]^{\beta}}$$

$\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{o_i\}$

enlever de $Candidats$ chaque objet qui viole des contraintes de ressources

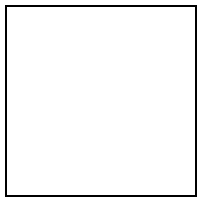
[redacted]

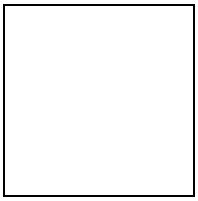
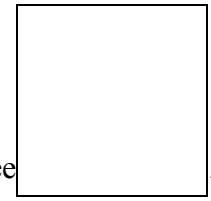
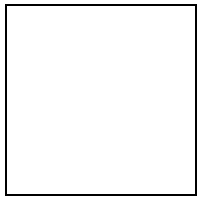
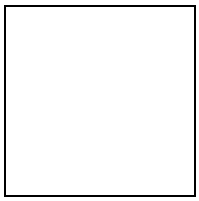
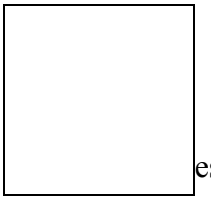
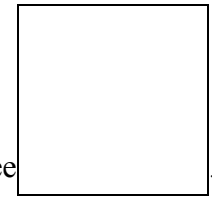
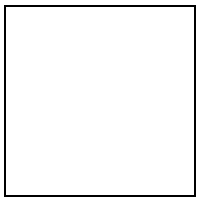
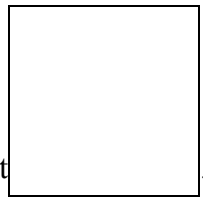
Fig. 5.1 - Algorithme ACO pour le MKP

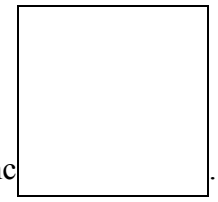
5.2.2 Définition des composants phéromonaux

Les fourmis de l'algorithme Ant-Knapsack évoluent et déposent de la phéromone sur le graphe complet G = (V,E) tel que V est l'ensemble des objets. Cet algorithme s'instancier en trois versions différentes en fonction des composants du graphe sur lesquels les fourmis déposent des traces de phéromone.

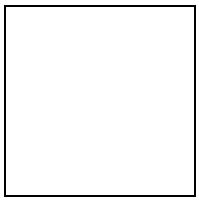
- Dans Vertex-AK, les fourmis déposent la phéromone sur les sommets V du graphe. La quantité de phéromone sur un objet [redacted] est notée [redacted].

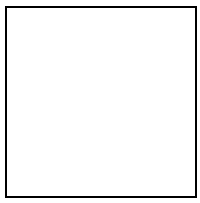
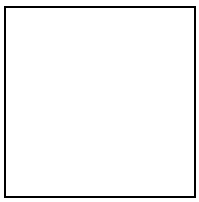
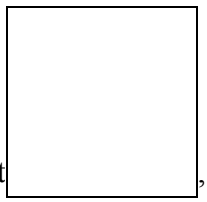
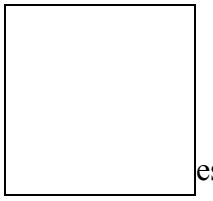
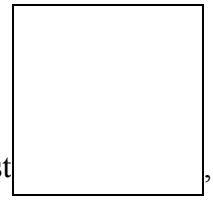
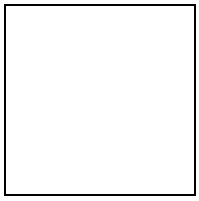
Cette quantité représente la « désirabilité » de choisir l'objet  lors de la construction d'une solution.

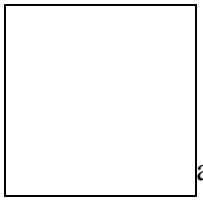
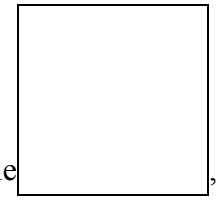
- Dans Path-AK, les fourmis déposent de la phéromone sur les couples de sommets sélectionnés consécutivement, i.e., sur les arcs du graphe. La quantité de phéromone sur un arc  est notée  . Cette quantité représente la « désirabilité » de choisir l'objet  juste après avoir choisi l'objet  lors de la construction d'une solution. Il est à noter que dans ce cas le graphe est orienté.
- Dans Edge-AK, les fourmis déposent la phéromone sur les paires de sommets sélectionnés dans une même solution, i.e., sur les arêtes E du graphe. La quantité de phéromone sur une arête  est notée  . Cette quantité représente la désirabilité de choisir un objet  lors de la construction d'une solution qui contient déjà l'objet  .

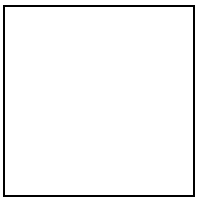
Il est à noter que, dans ce cas le graphe est non orienté, et donc  .

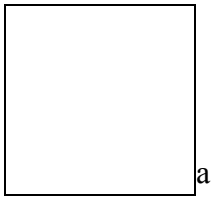
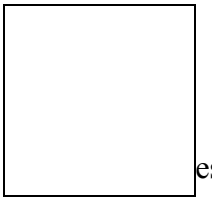
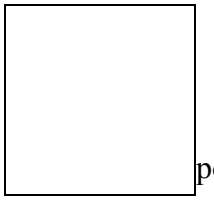
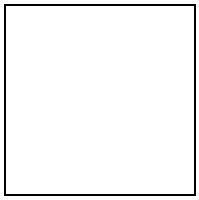
5.2.3 Définition du facteur phéromonal

Le facteur phéromonal  traduit l'expérience passée de la colonie et est utilisé dans la probabilité de transition des fourmis pour les guider vers des zones de l'espace de recherche prometteuses. Ce facteur phéromonal dépend de la quantité de phéromone déposée sur les composants phéromonaux du graphe :

- Dans Vertex-AK, il dépend de la quantité déposée sur l'objet candidat, i.e., 
- Dans Path-AK, il dépend de la quantité présente sur l'arc connectant le dernier objet ajouté à la solution partielle  et l'objet candidat  , i.e., si le dernier objet ajouté dans  est  , 

- Dans Edge-AK, il dépend de la quantité déposée sur les arêtes connectant l'objet candidat  avec les différents objets présents dans la solution partielle  , i.e.,



Notons que ce facteur phéromonal peut être calculé de façon incrémentale : lorsque le premier objet  a été sélectionné, le facteur phéromonal  est initialisé à  pour chaque objet candidat  ; ensuite, à chaque fois qu'un nouvel objet o_i est ajouté à la solution courante S_k, le facteur phéromonal $\tau_{S_k}(o_j)$ de chaque objet candidat o_j est incrémenté de $\tau(o_i, o_j)$.

5.2.4 Mise à jour de la phéromone

Après que toutes les fourmis aient fini la construction de leurs solutions, les traces de phéromone sont mises à jour. Cette mise à jour s'effectue en deux étapes. Dans une première étape, toutes les traces de phéromone sont diminuées, pour simuler l'évaporation, en multipliant chaque composant phéromonal par un ratio de persistance $(1-p)$ tel que $0 \leq p \leq 1$.

Notons que dans Vertex-AK, les composants phéromonaux sont associés aux objets de sorte que l'étape d'évaporation s'effectue en $O(n)$ tandis que dans Path-AK et Edge-AK, les

composants phéromonaux sont associés aux couples d'objets de sorte que l'étape d'évaporation s'effectue en $O(n^2)$.

Dans un deuxième temps, la meilleure fourmi du cycle dépose de la phéromone. Plus précisément, soit $S_k \in \{S_1,...,S_{nbAnts}\}$ la meilleure solution (celle ayant un profit maximal, les ex-aequo étant départagés aléatoirement) construite durant le cycle courant et S_{best} la meilleure solution construite depuis le début de l'exécution (y compris le cycle courant).
La quantité de phéromone déposée par la fourmi k est inversement proportionnelle à la différence de profit entre S_k et S_{best}, i.e., elle est égale à $1/(1 + profit(S_{best}) - profit(S_k))$.
Cette quantité de phéromone est déposée sur les composants phéromonaux de S_k, i.e.,

- dans Vertex-AK, elle est déposée sur chaque sommet de S_k,
- dans Path-AK, elle est déposée sur les arcs du chemin visité par la fourmi k lors de la construction de S_k, i.e., sur tout couple de sommets (o_i, o_j) tel que o_j a été ajouté dans S_k juste après o_i.
- dans Edge-AK, elle est déposée sur chaque arête reliant deux sommets différents de S_k, i.e., sur la clique définie par S_k.

Comme pour la première étape, la complexité en temps de cette deuxième étape dépend de la version considérée : pour Vertex-AK et Path-AK elle s'effectue en $O(|S_k|)$ tandis que pour Edge-AK elle s'effectue en $O(|S_k|^2)$

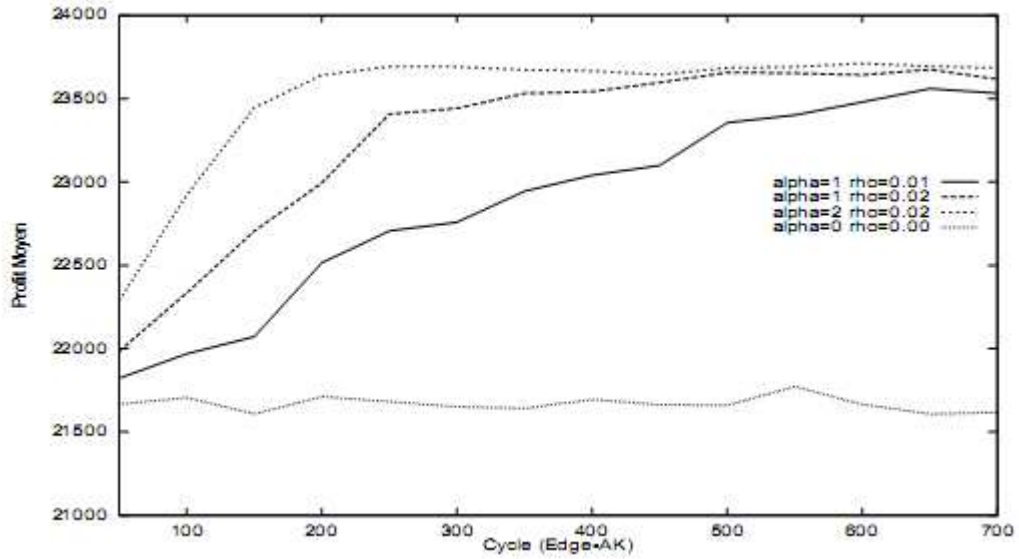
5.3 Influence des paramètres α et ρ sur la résolution

Quand on résout un problème d'optimisation combinatoire avec une approche heuristique, il s'agit de trouver un bon compromis entre deux objectifs relativement duaux : d'une part il s'agit d'intensifier la recherche autour des zones de l'espace de recherche les plus prometteuses, qui sont généralement proches des meilleures solutions trouvées ; d'autre part il s'agit de diversifier la recherche et favoriser l'exploration afin de découvrir de nouvelles et si possible meilleures zones de l'espace de recherche. Le comportement des fourmis par rapport à cette dualité entre intensification et diversification peut être influencé en modifiant les valeurs des paramètres. En particulier, la diversification peut être augmentée soit en diminuant la valeur du poids du facteur phéromonal α (de sorte que les fourmis deviennent

moins sensibles aux traces phéromonales), soit en diminuant le taux d'évaporation ρ (de sorte que la phéromone s'évapore plus doucement et les écarts d'une trace à l'autre évoluent plus doucement). Lorsque l'on augmente ainsi la capacité exploratoire des fourmis, on trouve généralement de meilleures solutions, mais en contrepartie ces solutions sont plus longues à trouver.

Nous avons pu constater que cette influence des paramètres α et ρ sur le comportement des fourmis est identique pour les trois versions proposées de Ant-Knapsack. Nous l'illustrons dans la Figure 5.2 pour Edge-AK et Vertex-AK. Dans cette figure, chaque courbe trace l'évolution du profit moyen quand le nombre de cycles augmente pour une affectation donnée de α et ρ (moyenne sur 10 exécutions). Les autres paramètres ont été affectés à $\beta = 5$, $nbAnts = 30$, $\tau_{\min} = 0.01$, et $\tau_{\max} = 6$. Cette figure montre que quand on augmente l'intensification, en choisissant des valeurs telles que $\alpha = 2$ et $\rho = 0.02$, Edge-AK trouve rapidement de bonnes solutions mais stagne plus vite sur des solutions légèrement moins bonnes. A l'inverse, en choisissant des valeurs pour α et ρ qui favorisent l'exploration, telles que $\alpha = 1$ et $\rho = 0.01$, les fourmis trouvent de meilleures solutions en fin d'exécution, mais elles ont besoin de plus de cycles pour converger sur ces solutions.

Notons finalement que lorsque la phéromone n'est pas utilisée du tout, i.e., quand $\alpha = 0$ et $\rho = 0$, de sorte que l'algorithme se comporte comme un algorithme glouton classique, les solutions trouvées sont très nettement moins bonnes que lorsque la phéromone est utilisée.



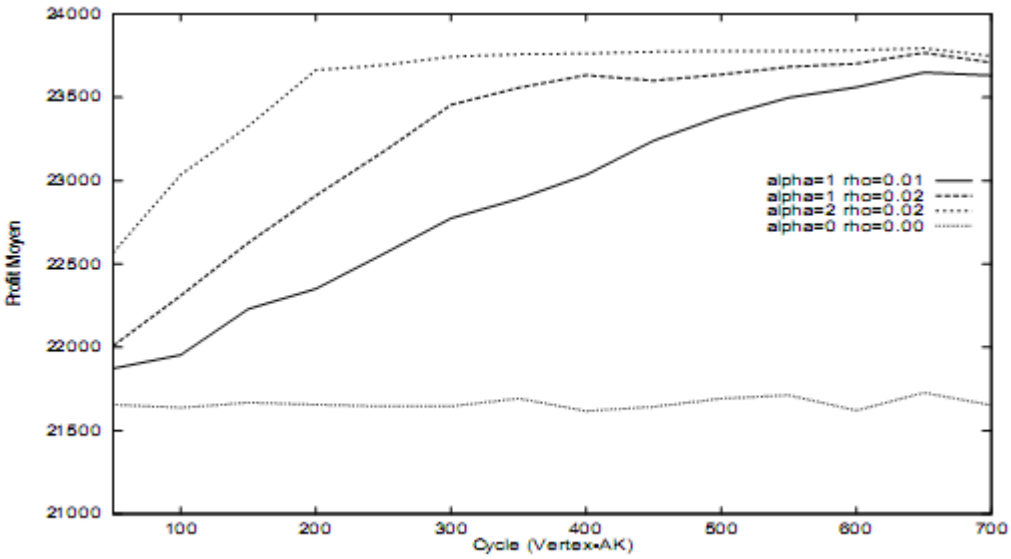


Fig. 5.2 - Influence de α et ρ sur la qualité des solutions trouvées par Edge-AK et Vertex-AK pour une instance du MKP ayant 100 objets et 5 contraintes de ressource

5.4 Influence des traces de phéromone sur la similarité des solutions calculées

Pour expliciter l'influence de la phéromone sur la capacité des fourmis à explorer l'espace de recherche, nous proposons dans cette section de mesurer la similarité des solutions construites durant l'exécution. Plusieurs mesures de diversité ou de similarité ont été introduites pour les approches évolutionnaires, le maintien de la diversité de la population étant un point clé pour éviter une convergence prématurée et une stagnation de ces algorithmes.

Pour mesurer l'effort de diversification de Ant-Knapsack pendant l'exécution, nous proposons de calculer le taux de similarité. Ce taux correspond à la mesure de diversité de la population introduite pour les approches génétiques [125]. Plus précisément, le taux de similarité d'un ensemble de solutions S est défini par le nombre moyen d'objets qui sont partagés par n'importe quelle paire de solutions dans S, divisé par le nombre moyen d'objets sélectionnés dans les solutions de S. Ainsi, ce taux est égal à un si toutes les solutions de S sont identiques, tandis qu'il est égal à zéro si l'intersection de chaque paire de solutions de S est vide.

Il est à noter que ce ratio peut être calculé très rapidement en maintenant un tableau freq tel que, pour chaque nœud $v_i \in V$, freq[i] est égal au nombre de solution de S qui ont sélectionné le noeud v_i. Dans ce cas, le ratio de similarité de S est égal à

$\frac{\sum_{v_i \in V} (freq\ [i].(\ frreq\ [i] - 1))}{(|S| - 1).\sum_{S_k \in S} |S_k|}$ et il peut être calculé facilement de manière incrémentale en construisant les solutions.

Dans les algorithmes ACO, ce sont les traces de phéromone qui dirigent les fourmis vers les zones « prometteuses » de l'espace de recherche. La figure 5.3 montre l'influence de ces traces sur la similarité des solutions construites durant l'exécution de l'algorithme Ant-Knapsack, pour les deux variantes Path-AK et Edge-AK, et pour différentes valeurs des paramètres α et ρ qui déterminent l'influence de la phéromone sur le comportement des fourmis. Dans cette figure, chaque courbe trace l'évolution du ratio de similarité des solutions construites quand le nombre de cycles augmente pour une affectation donnée de α et ρ (moyenne sur 10 exécutions). Les autres paramètres ont été affectés à $\beta = 5$, nbAnts = 30, $\tau_{\min} = 0.01$, et $\tau_{\max} = 6$. Ces courbes mettent en évidence trois phases dans les exécutions de Ant-Knapsack.

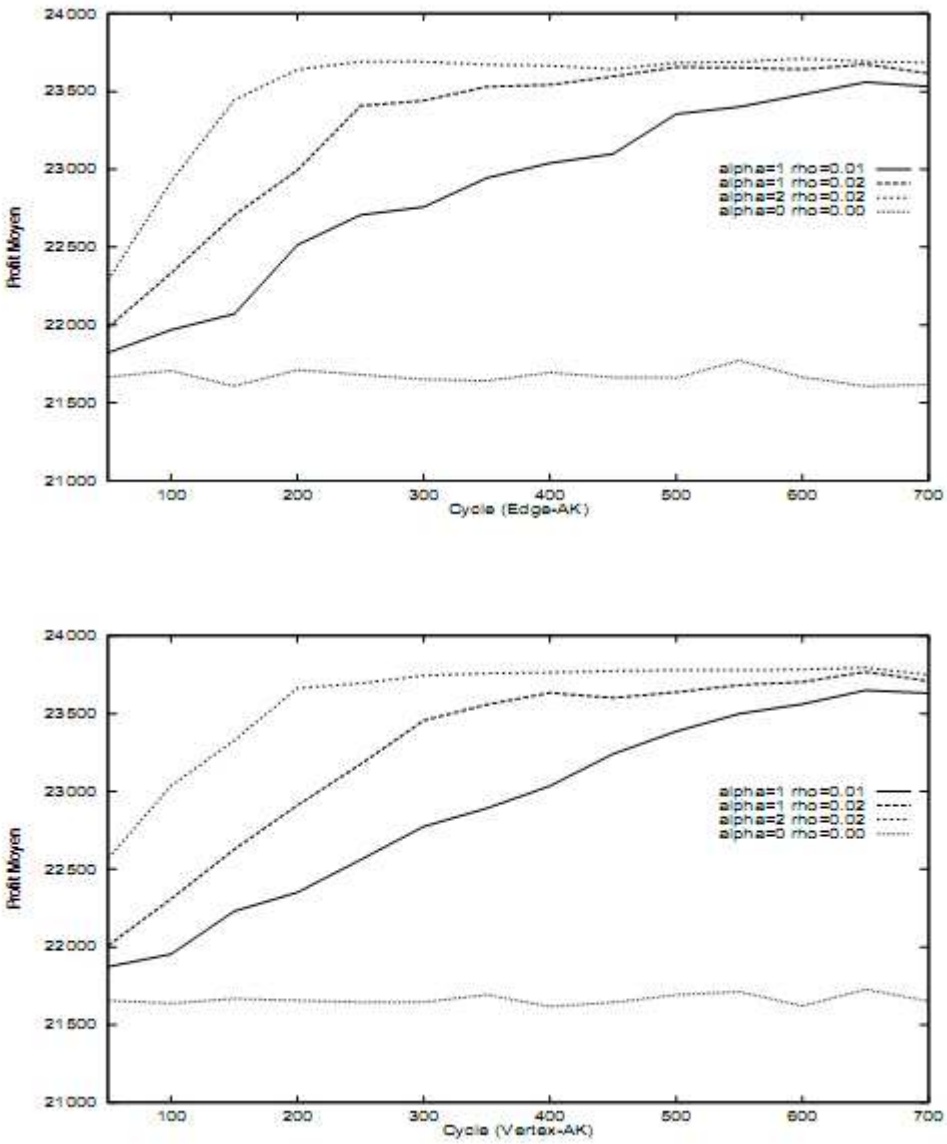


Fig. 5.3 - Influence de α et ρ sur la similarité des solutions trouvées par Edge-AK et Path-AK pour une instance du MKP ayant 100 objets et 5 contraintes de ressource.

Durant les premiers cycles, on observe une phase d'exploration, où les fourmis calculent des solutions assez différentes (les solutions construites ont 42% d'objets en commun en moyenne). La durée de cette phase d'exploration dépend du paramétrage considéré : elle est d'autant plus longue que l'on réduit l'influence de la phéromone en diminuant α et ρ. La durée de cette phase d'exploration dépend aussi de la stratégie phéromonale considérée : on observe qu'elle est plus longue pour Path-AK que pour Edge-AK. Notons que cette phase d'exploration initiale est accentuée par la stratégie du MAX −MIN Ant System, qui impose des bornes minimales et maximales $\tau_{\min}$ et $\tau_{\max}$ aux traces de phéromone, et initialise ces traces à $\tau_{\max}$ au début de l'exécution : ainsi, après *n* cycles de Ant-knapsack, la trace de

phéromone τ_i sur un composant phéromonal (arête ou sommet) *i* est telle que $(1-\rho)\cdot \tau_{\min} \leq \tau_i \leq \tau_{\max}$, et comme $(1-\rho)$ est généralement très proche de 1, les traces de phéromone ont des valeurs très similaires durant les premiers cycles et n'influencent pas significativement le calcul des probabilités de transition.

On observe ensuite une deuxième phase de *convergence*, où la similarité des solutions construites augmente progressivement, mettant en évidence le fait que les fourmis sont de plus en plus influencées par la phéromone et intensifient l'effort de recherche autour d'une zone plus petite de l'espace de recherche : la similarité des solutions construites augmente de 42% à près de 90%. Notons que l'augmentation de la similarité est d'autant plus rapide que l'influence de la phéromone est forte.

On observe enfin une troisième phase de *stagnation*, où la similarité des solutions construites se stabilise autour d'une valeur assez élevée (entre 80% et 90%), montrant que les fourmis se sont focalisées sur une toute petite zone de l'espace de recherche qu'elles explorent intensément.

5.5 Expérimentations et résultats

Nous présentons dans cette section les résultats des trois variantes de l'algorithme Ant-Knapsack, i.e., Vertex-AK, Path-AK et Edge-AK. Nous comparons aussi ces résultats avec ceux de deux algorithmes ACO qui ont été proposés dans la littérature et qui utilise chacun une stratégie phéromonale différente. L'algorithme de Leguizemon et Michalewicz [123] utilise la même stratégie de Vertex-AK et l'algorithme de Fidanova [126] qui utilise la même stratégie que Path-AK. Nous comparons enfin ces résultats avec deux approches de l'état de l'art sur le sujet qui trouvent parmi les meilleurs résultats connus pour ces instances.

5.5.1 Ensemble de tests et conditions d'expérimentation

Nous avons considéré des instances larges de MKP OR-Library accessibles à partir du site http://www.tik.ee.ethz.ch/sop/download/supplementary/testProblemSuite/.Ces instances sont groupées en 5 classes avec $m \in \{5,10,30\}$ et $n \in \{100,250\}$. Dans la suite, ces classes sont notées *m.n*. Chaque classe contient 30 instances groupées en 3 sous-classes de 10 instances en fonction d'un ratio de dureté *rd : rd = 0, 25* pour les 10 premières instances, *rd = 0, 5* pour les 10 instances suivantes, et *rd = 0, 75* pour les 10 dernières instances.

Dans toutes les expérimentations, α est mis à 1, β à 5, ρ à 0.01, le nombre de fourmis à 30 et les bornes de phéromone $\tau_{\min}$ et $\tau_{\max}$ à 0.01 et 6. Le nombre de cycles est limité à 2000 cycles.

5.5.2 Comparaison des algorithmes Vertex-AK, Path-AK et Edge-AK

Le tableau 5.1 compare la qualité des résultats obtenus par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées. Chaque classe contient 10 instances, et chaque algorithme a été exécuté 10 fois sur chaque instance, de sorte que chaque résultat est une moyenne (ou un écart-type) sur 100 exécutions. Dans ce tableau, chaque ligne donne les résultats pour les 10 instances de la classe *m.n* ayant le ratio de dureté *rd*. Pour chacune de ces classes et pour chaque algorithme considéré, le tableau donne le profit trouvé (moyenne et écart-type sur 10 exécutions pour chacune des 10 instances de la classe).

Ce tableau montre que les deux algorithmes Vertex-AK et Edge-AK trouvent des résultats très nettement meilleurs que ceux trouvés par Path-AK, pour toutes les classes d'instances considérées. Ainsi, la stratégie consistant à déposer de la phéromone sur des objets consécutivement sélectionnés guide les fourmis de manière moins profitable que les deux autres stratégies.

En comparant Vertex-AK et Edge-AK, nous constatons que les solutions trouvées par Edge-AK sont meilleures pour 10 classes d'instances et inférieures pour les 5 autres classes. Notons que ces résultats dépendent de la taille des instances (par rapport au nombre de variables et de contraintes) : pour les instances des classes 5.100, 5.250 et 10.100, Edge-AK est clairement meilleur que Vertex-AK tandis que pour les instances des classes 10.250 et 30.100 cette tendance s'inverse.

Le tableau 5.2 compare les temps d'exécution et le nombre de cycles effectués par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées. Dans ce tableau, chaque ligne donne les résultats pour les 10 instances de la classe *m.n* ayant le ratio de dureté *rd*. Pour chacune de ces classes et pour chaque algorithme considéré, le tableau donne le temps (en secondes) et le nombre de cycles nécessaires pour trouver la solution (moyenne (Moy) et écart-type (Ec) sur 10 exécutions de chaque instance de la classe). On constate que les résultats pour les trois versions sont du même ordre de grandeur. Path-AK effectue légèrement moins de cycles et est un peu plus rapide que les deux autres versions, mais cela s'explique probablement par le fait qu'il trouve de bien moins bonnes solutions : pour Path-

AK, la phéromone n'est pas capable de guider la recherche vers de bonnes zones de l'espace de recherche.

On constate que les nombres de cycles effectués par Vertex-AK et Edge-AK sont très similaires. Ce nombre dépend essentiellement du nombre de variables n, tandis que le nombre de contraintes m et la dureté de ces contraintes *rd* ne semblent pas avoir une influence directe. Enfin, on constate qu'un cycle de Vertex-AK est effectué légèrement plus rapidement qu'un cycle de Edge-AK. Par exemple, pour la classe 30.100 avec *rd = 0, 25*, les deux algorithmes ont besoin en moyenne de 810 cycles pour converger ; Vertex-AK met 234 secondes pour cela tandis que Edge-AK met 316 secondes. Cette différence, qui est d'autant plus sensible que le nombre de variables n augmente, s'explique par le fait que l'évaporation est en $O(n^2)$ pour Edge-AK alors qu'elle est en $O(n)$ pour Vertex-AK.

Tab. 5.1 - Comparaison de la qualité des solutions trouvées par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérés.

m.n	rd	Vertex-AK		Path-AK		Edge-AK	
		Moyenne	Ecart	Moyenne	Ecart	Moyenne	Ecart
5.100	0,25	24192,0	33,13	24139,3	149,52	**24197,2**	37,52
5.100	0,50	43243,3	24,89	43137,9	137,11	**43246,4**	35,87
5.100	0,75	60470,5	25,37	60435,7	68,86	**60471,0**	28,35
5.250	0,25	**60335,9**	56,09	59110,6	911,77	60334,9	50,16
5.250	0,50	109203,4	39,09	107818	554,38	**109231,6**	47,97
5.250	0,75	151536,6	36,93	150578,6	152,68	**151536,9**	37,86
10.100	0,25	22553,0	60,97	22418,2	193,32	**22572,2**	61,81
10.100	0,50	42641,0	50,60	42383,0	192,37	**42658,5**	60,97
10.100	0,75	59540,0	34,71	59464,9	89,42	**59555,6**	44,70
10.250	0,25	**58847,6**	124,66	57367,3	593,35	58826,0	93,55
10.250	0,50	**108600,7**	109,50	106834,2	506,34	108594,1	78,03
10.250	0,75	151272,5	51,03	150066,9	170,66	**151276,0**	55,46
30.100	0,25	**21618,1**	73,33	21411,4	213,22	21608,4	75,68
30.100	0,50	**41406,1**	69,05	41013,0	228,18	41403,3	68,53
30.100	0,75	59172,5	40,56	59049,9	111,09	**59173,2**	36,67
Moyenne		67642,2	55,33	67015,26	284,8	**67645,7**	54,21

Tab. 5.2 - Comparaison des temps d'exécution de Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérés.

m.n	rd	Vertex-AK				Path-AK				Edge-AK			
		Temps		Nb cycles		Temps		Nb cycles		Temps		Nb cycles	
		Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart
5.100	0,25	20	25	619	219	24	9	759	296	25	19	496	209
5.100	0,50	38	50	585	215	44	18	739	321	47	39	490	157
5.100	0,75	37	49	381	142	63	28	713	315	56	60	333	125
5.250	0,25	327	111	1062	357	182	97	720	384	343	89	1237	321
5.250	0,50	715	184	1115	287	438	238	779	426	807	202	1321	335
5.250	0,75	1044	304	1038	276	745	392	828	437	1185	328	1197	333
10.100	0,25	28	12	751	335	26	12	756	366	21	11	614	325
10.100	0,50	52	22	630	261	49	25	699	351	43	23	508	297
10.100	0,75	56	24	489	224	79	41	766	401	37	24	358	242
10.250	0,25	373	172	1054	364	267	165	806	428	413	129	1167	360
10.250	0,50	813	276	1151	390	489	274	743	415	971	235	1315	317
10.250	0,75	1232	396	1132	364	840	514	812	499	1364	360	1230	318
30.100	0,25	234	250	808	325	93	81	774	309	316	248	810	372
30.100	0,50	284	291	749	302	113	56	769	386	400	213	797	31
30.100	0,75	262	292	515	188	162	78	853	428	437	267	587	251
Moyenne		368	164	805	283	241	135	768	384	431	150	831	285

5.5.3 Comparaison avec d'autres algorithmes ACO

L'algorithme Ant-Knapsack s'instancie en trois versions Vertex-AK, Path-AK et Edge-AK en fonction de la stratégie phéromonale choisie. L'instance Edge-AK correspond à l'algorithme décrit dans [124]. Les instances Vertex-AK et Path-AK adoptent les stratégies phéromonales respectivement proposées dans [123] et [126], mais présentent quelques différences par ailleurs. Nous comparons dans cette section les résultats obtenus par ces différents algorithmes ACO pour le MKP.

Comparaison de Vertex-AK avec l'algorithme de Leguizamon et Michalewizc

L'algorithme Vertex-AK utilise la même stratégie phéromonale que l'algorithme proposé par Leguizamon et Michalewizc dans [123]. Ces deux algorithmes diffèrent cependant sur les deux points suivants :

- dans l'algorithme de Leguizamon et Michalewizc, la quantité de phéromone déposée est égale au profit de la solution construite tandis que dans Vertex-AK elle est inversement proportionnelle à l'écart de profit avec la meilleure solution trouvée ;
- l'algorithme de Leguizamon et Michalewiz est basé sur le schéma ACO « Ant-System » proposé initialement dans [127] tandis que Vertex-AK est basé sur le schéma ACO « MAX-MIN Ant System » proposé dans [122].

Pour les deux algorithmes, les paramètres α et β, qui déterminent l'importance relative du facteur heuristique et du facteur phéromonal, ont été fixés aux mêmes valeurs (i.e., $\alpha = 1$ et $\beta = 5$). En revanche, les résultats présentés dans [123] ont été obtenus avec un taux d'évaporation ρ fixé à 0.3 (au lieu de 0.01 pour Vertex-AK), un nombre de fourmis égal au nombre d'objets, i.e., n (au lieu de 30 pour Vertex-AK) et un nombre de cycles fixé à 100 (au lieu de 2000 pour Vertex-AK).
Sur les instances des classes 5.100 et 10.100 dont le ratio de dureté est rd = 0, 25, l'algorithme de Leguizamon et Michalewiz obtient des profits de 24144 et 22457 respectivement (moyenne sur 10 exécutions pour chacune des 10 instances de la classe considérée).
Si l'on compare ces résultats avec ceux de Vertex-AK (qui obtient 24192 et 22553 sur ces deux classes), on constate que Vertex-AK obtient de meilleurs résultats. Notons toutefois que sur ces instances chaque exécution de l'algorithme de Leguizamon et Michalewiz calcule $100 * 100 = 10000$ solutions tandis que Vertex-AK en calcule $30 * 2000 = 60000$.

Comparaison de Path-AK avec l'algorithme de Fidanova
L'algorithme Path-AK utilise la même stratégie phéromonale que l'algorithme proposé par Fidanova dans [126]. Ces deux algorithmes diffèrent cependant sur les mêmes points que Vertex-AK par rapport à l'algorithme de Leguizamon et Michalewizc, i.e., sur la quantité de phéromone déposée et sur le schéma ACO considéré. De plus, l'algorithme de Fidanova renforce la meilleure solution trouvée par une quantité de phéromone additionnelle.
L'information heuristique dans cet algorithme est calculée de façon statique, i.e., sans prendre en compte les quantités de ressources déjà consommées par la solution en cours de construction.

Sur les instances de la classe 5.100 avec les valeurs du ratio de dureté *rd = 0, 25 rd = 0, 5* et *rd = 0, 75,* l'algorithme de Fidanova obtient des profits moyens respectivement de 23939,43037 et 60373. Si l'on compare ces résultats avec ceux de Path-AK (qui obtient 24139,43137 et 60435), on constate que Path-AK obtient de meilleurs résultats. Notons toutefois que sur ces instances chaque exécution de l'algorithme de Fidanova calcule $200 * 100 = 20000$ solutions tandis que Path-AK en calcule $30 * 2000 = 60000$.

Comparaison de Edge-AK avec les deux algorithmes ACO
On compare maintenant les algorithmes Edge-AK, l'algorithme de Leguizamon et Michalewizc et celui de Fidanova, qui utilise chacun une stratégie phéromonale différente. Nous trouvons que Edge-AK trouve des résultats qui sont nettement meilleurs que ceux trouvés par l'algorithme de Fidanova sur toutes les instances testées : pour la classe d'instances 5.100 l'algorithme de Fidanova trouve un profit de 42449 (moyenne des 30 instances testées) tandis que Edge-AK trouve un profit de 42638. Edge-AK trouve aussi des résultats meilleurs que ceux reportés par l'algorithme de Leguizamon et Michalewizc sur la plupart des instances testées : sur les instances des classes 5.100 et 10.100 dont le ratio de dureté est rd = 0, 25, l'algorithme de Leguizamon et Michalewiz obtient des profits de 24144 et 22457, alors que Edge-AK trouve des profits de 24197 et 60471 respectivement.

5.5.4 Comparaison avec d'autres approches

On compare maintenant Edge-AK, qui est la version de Ant-Knapsack qui obtient les meilleurs résultats en moyenne avec une approche proposée par Chu et Beasley [128] et une approche proposée par Gottlieb [129]. Ces deux approches combinent un algorithme génétique avec des techniques de réparation (pour transformer une solution violant des contraintes en une solution cohérente) et des techniques de recherche locale (pour améliorer la qualité des solutions cohérentes). Dans les deux approches la méthode de remplacement utilisée est une méthode « steady state », la méthode de sélection est le tournoi binaire standard. Les deux approches ont une population de 100 individus et stoppent une exécution lorsque 106 solutions différentes ont été générées. Les différences entre les deux approches concernent essentiellement l'opérateur de croisement et la procédure de génération de la population initiale. Il est à noter que les résultats trouvés par ces deux approches pour la plupart des instances considérées sont des valeurs optimales (pour les instances avec des valeurs optimales connues).

Le tableau 5.3 compare Ant-knapsack avec ces deux approches sur 6 classes de 10 instances du MKP. Pour chaque classe, on donne l'écart moyen entre la solution sur les réels et la solution trouvée par Edge-AK, Chu et Beasley (CB), et Gottlieb (G). Pour Edge-AK et Gottlieb, on reporte également le nombre moyen de solutions construites pour trouver la meilleure solution. Dans ce tableau, la qualité des solutions est mesurée par l'écart (en pourcentage) entre le profit de la meilleure solution trouvée et la valeur optimale de la relaxation du problème linéaire sur les réels, i.e., $Ecart = 1 - Best / opt^{LP}$ où opt^{LP} est la valeur optimale du problème linéaire sur les réels et Best est la meilleure solution trouvée par l'algorithme considéré.

Si l'on considère la qualité des solutions trouvées, on remarque que Edge-AK obtient les mêmes résultats que CB sur quatre classes et les mêmes résultats que G sur trois classes. Il obtient de meilleurs résultats que G sur la classe 10.100 avec *rd = 0, 5*. Sur les deux autres classes, il obtient des résultats légèrement moins bons. Notons toutefois que les deux algorithmes génétiques considérés utilisent des techniques de recherche locale pour améliorer la qualité des solutions construites, ce qui n'est pas le cas de Edge-AK. De plus, quand on considère le nombre de solutions construites, on remarque que Edge-AK construit au plus soixante mille solutions tandis que les deux algorithmes génétiques en construisent au plus un million. Le tableau 3.3 montre que G construit toujours plus de solutions que Ant-Knapsack.

Tab. 5.3 - Comparaison de Ant-knapsack avec deux approches évolutionnaires sur 6 classes de 10 instances du MKP.

Classe			CB	G		Edge-AK	
M	n	rd	Ecart	écart	#sol	écart	#sol
5	100	0.25	0.99	0.99	74595	0.99	15736
5	100	0.50	0.45	0.45	37353	0.47	16053
5	100	0.75	0.32	0.32	40690	0.32	11131
Moyenne pour 5.100			0.587	0.587	50879	0.59	14306
5	100	0.25	1.56	1.60	190979	1.69	17874
5	100	0.50	0.79	0.81	109036	0.79	17147
5	100	0.75	0.48	0.48	53528	0.48	15137
Moyenne pour 10.100			0.943	0.965	117848	0.96	16919

5.6 Conclusion

Nous avons présenté dans ce chapitre un algorithme ACO générique pour résoudre le problème du sac à dos multidimensionnel. Nous avons présenté trois versions différentes de cet algorithme suivant les composants sur lesquels les traces de phéromone sont déposées : Vertex-AK où la phéromone est déposée sur les objets, Path-AK où la phéromone est déposée sur les couples d'objets sélectionnés consécutivement, et Edge-AK où la phéromone est déposée sur les paires d'objets sélectionnés dans une même solution. Un objectif principal était de comparer ces trois stratégies phéromonales, indépendamment des détails d'implémentation et du schéma ACO considéré.

Les expérimentations ont montré que Path-AK obtient de bien moins bons résultats que Vertex-AK et Edge-AK : pour ce problème, l'ordre dans lequel les objets sont sélectionnés n'est pas significatif, et il n'est clairement pas intéressant de ne considérer que le dernier objet sélectionné pour choisir le prochain objet à entrer dans le sac à dos. Les expérimentations ont également montré que Edge-AK obtient de meilleurs résultats que Vertex-AK sur deux tiers des classes d'instances considérées. Cependant, les différences de résultats entre ces deux versions ne sont pas aussi importantes qu'avec Path-AK. De plus, Edge-AK prend plus de temps CPU que Vertex-AK lors de l'exécution. Ceci est dû à la phase de mise à jour de phéromone qui est de complexité quadratique pour Edge-AK alors qu'elle est de complexité linéaire pour Vertex-AK.

Comme nous l'avons souligné dans l'introduction de ce chapitre, nous pensons que cette étude expérimentale de différentes stratégies phéromonales dépasse le cadre du MKP, et peut être utile pour la résolution d'autres problèmes de « recherche d'un meilleur sous-ensemble » à l'aide de la métaheuristique ACO.

En comparant Edge-AK avec d'autres algorithmes ACO de l'état de l'art, qui utilisent des stratégies phéromonales différentes, on trouve qu'il donne de meilleurs résultats pour toutes les instances considérées.

Lorsque l'on compare les résultats obtenus par Edge-AK avec d'autres approches de l'état de l'art, qui trouvent parmi les meilleurs résultats connus pour les instances testées, on constate qu'ils sont globalement compétitifs même s'ils sont légèrement moins bons sur quelques instances, puisqu'ils retrouvent la plupart de ces résultats. Notons toutefois qu'Edge-AK n'utilise pas de technique de recherche locale pour améliorer les solutions construites par les fourmis.

Chapitre 6 : Optimisation par colonies de fourmis pour des problèmes multi-objectifs

6.1 Optimisation par colonies de fourmis

6.1.1 Introduction

La métaheuristique d'optimisation par colonies de fourmis a été initialement introduite par Dorigo, Maniezzo et Colorni [130, 131] et a été inspirée par les études sur le comportement des fourmis réelles effectuées par Deneubourg et al [127].

A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le problème du voyageur de commerce en proposant le premier algorithme ACO : 'Ant System' (AS) [131]. Par la suite, un nombre considérable d'applications de ACO a été proposé telles que l'affectation quadratique [132], le routage des véhicules [133], le problème de satisfaction de contraintes [134],...

Dans la section suivante nous expliquons l'analogie biologique à partir de laquelle ACO a été inspirée.

6.1.2 Analogie biologique

La métaheuristique ACO a été inspirée, essentiellement, par les études sur le comportement des fourmis réelles effectuées par Deneubourg et al [127]. L'un des problèmes étudiés était de comprendre comment des insectes, comme les fourmis, peuvent trouver le chemin le plus court du nid à la source de nourriture et le chemin de retour.

Il a été trouvé que le moyen utilisé pour communiquer l'information entre les fourmis qui cherchent des chemins, est le dépôt de traces de phéromone,i.e., une substance chimique que les fourmis arrivent à détecter.

En se déplaçant, une fourmi dépose de la phéromone marquant ainsi le chemin par une trace de cette substance. Tandis qu'une fourmi isolée se déplace aléatoirement, une fourmi qui rencontre une trace de phéromone déjà déposée peut la détecter et décider de la suivre avec une probabilité proportionnelle à l'intensité de la trace, et renforce ainsi cette trace avec sa propre phéromone.

Le comportement collectif qui émerge est une forme de comportement auto-catalytique où plus les fourmis suivent une trace, plus cette trace devient attirante : C'est le principe de stigmergie.

Ce processus peut être illustré par l'exemple de la figure 6.1. Dans la figure 6.1.a, des fourmis se déplacent dans un réseau qui connecte une source de nourriture (A) à un nid (E). Supposons que les fourmis prennent une unité de temps pour une distance de longueur 1 et que à chaque unité de temps, 30 fourmis sortent du nid et de la source de nourriture, respectivement. Les premières fourmis qui arrivent aux position B et D doivent choisir d'aller vers C (avec une longueur totale du chemin de 3) ou vers H (avec une longueur totale du chemin de 4). Supposons qu'à l'instant t=0, 30 fourmis sont au point B (et 30 au point D). Comme il n'existe pas de trace de phéromone sur les deux chemins, elles vont choisir une des deux alternatives avec la même probabilité. Donc 15 fourmis décident d'aller à C et les 15 autres fourmis décident d'aller à H. A t=1, les 30 prochaines fourmis sortant de B (et de D) peuvent détecter les traces de phéromone (figure 6.1.c). La trace de phéromone sur BCD est deux fois plus intense que celle sur BHD, car 30 fourmis sont passées par BCD (15 de A et 15 de D), tandis que 15 fourmis seulement sont passées par BHD. C'est pourquoi maintenant 20 fourmis prennent BCD et 10 prennent BHD (de même 20 prennent DCB et 10 prennent DHB à partir du point D). Une fois encore, plus de phéromone est déposée sur le plus court chemin. Ce processus est répété à chaque unité de temps et ainsi les traces de phéromone sont renforcées. Grâce à ce processus auto-catalytique, toutes les fourmis vont très rapidement choisir le chemin le plus court.

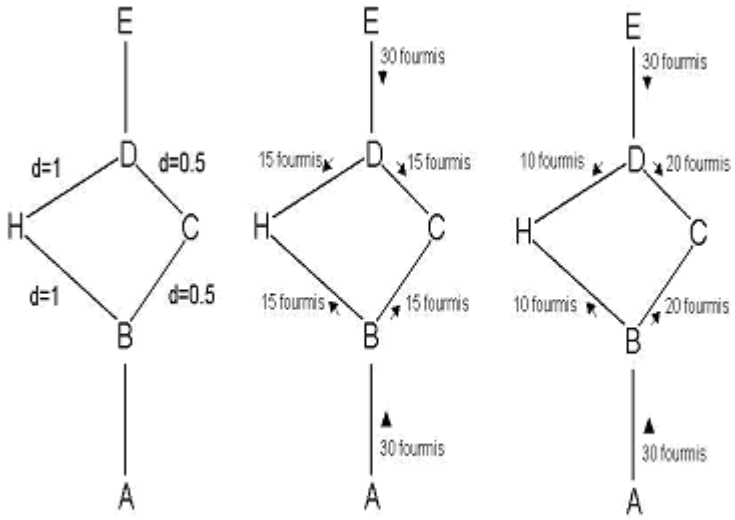


Fig 6.1. Comment les fourmis trouvent le plus court chemin

Pour transposer ce comportement à un algorithme général d'optimisation combinatoire, on fait une analogie entre l'environnement dans lequel les fourmis cherchent de la nourriture et l'ensemble des solutions admissibles du problème (i.e., l'espace de recherche du problème), entre la quantité ou la qualité de la nourriture et la fonction objectif à optimiser et enfin entre les traces et une mémoire adaptative.
Les fourmis artificielles dans les algorithmes ACO se comportent de la même manière.
Elles diffèrent des fourmis naturelles dans le fait qu'elles ont une sorte de mémoire, pour assurer la génération de solutions faisables. En plus, elles ne sont pas complètement aveugles, i.e. elles ont des informations sur leur environnement.
Nous allons décrire, dans ce qui suit, le développement historique de l'algorithme Ant System appliqué au problème de voyageur de commerce, ensuite nous définissons de façon plus générique la métaheuristique ACO.

6.1.3 Optimisation par colonie de fourmis et problème de voyageur de commerce

Le problème de voyageur de commerce (PVC) a fait l'objet de la première implémentation d'un algorithme de colonie de fourmis : le « Ant System » [130].

6.1.4 Ant System

Dans l'algorithme Ant System (AS), chaque fourmi est initialement placée sur une ville choisie aléatoirement, chacune possède une mémoire qui stocke la solution partielle qu'elle a construite auparavant. Initialement, la mémoire contient la ville de départ. Commençant à partir de cette ville, une fourmi se déplace itérativement d'une ville à une autre. Quand elle est à une ville *i*, une fourmi *k* choisit d'aller à une ville non encore visitée *j* avec une probabilité donnée par :

$$p_{ij}^{k}(t)=\frac{[\tau_{ij}(t)]^{\alpha}*[\eta_{ij}]^{\beta}}{\sum_{y\in N_i^k}[\tau_{iy}(t)]^{\alpha}*[\eta_{iy}]^{\beta}} \text{ si } j\in N_i^k \text{ ;0 sinon}$$

- $\tau_{ij}(t)$ est l'intensité de la trace de phéromone dans l'arête (*i, j*) à l'instant *t*,

- $\eta_{ij} = \frac{1}{d_{ij}}$ est une information heuristique à priori valable, où d_{ij} est la distance entre la ville *i* et la ville *j* ; l'idée étant d'attirer les fourmis vers les villes les plus proches,
- α, β sont deux paramètres qui déterminent l'influence relative de la trace de phéromone et de l'information heuristique,
- N_i^k est le voisinage faisable de la fourmi *k* c'est à dire l'ensemble des villes non encore visitées par la fourmi *k* accessibles depuis la ville *i.*

La construction de solution se termine après que chaque fourmi ait complété un tour. C'est à dire après que chaque fourmi ait construit une séquence de longueur *n*. Ensuite, les traces de phéromone sont mises à jour. Dans AS, la mise à jour se fait, d'abord, en réduisant les traces de phéromone avec un facteur constant ρ (c'est l'évaporation de phéromone) et, ensuite, en permettant à chaque fourmi de déposer de la phéromone sur les arêtes qui appartiennent à son tour. Ainsi la formule de mise à jour de phéromone est comme suit :

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{nbAnts} \Delta\tau_{ij}^k$$

Avec $0 \leq \rho \leq 1$ et *nbAnts* le nombre de fourmis.

Le paramètre ρ est ainsi utilisé pour éviter l'accumulation illimitée de phéromone et permet à l'algorithme d'oublier les mauvaises décisions précédemment prises. Sur les arêtes qui n'ont pas été choisis par les fourmis la force associée va décroître rapidement avec le nombre d'itérations.

$\Delta\tau_{ij}^k$ est le montant de phéromone que la fourmi *k* dépose sur l'arête (*i,j*).

Il est défini par :

$$\Delta\tau_{ij}^k = \frac{Q}{L^k} \text{ si } (i,j) \in Tabou^k \text{ ; 0 sinon}$$

où L^k est la longueur du tour généré par la fourmi *k* et *Q* une constante de l'algorithme. Avec cette formule, les arêtes du tour le plus court recevront la plus grande quantité de phéromone. En général, les arêtes qui sont utilisés par plusieurs fourmis et qui appartiennent aux tours les plus courts recevront plus de phéromone et en conséquence seront plus favorisés dans les itérations futures de l'algorithme.

L'algorithme AS complet est décrit dans ce qui suit :

Algorithme Ant System()
Soient G=(V,E) le graphe de construction,
$\alpha, \beta, \rho, nbAnts, Q, \tau_{init}$ les paramètres de l'algorithme

1. Pour chaque arête $(i, j) \in E$ Faire $\tau_{ij}(0) \leftarrow \tau_{init}$, $t \leftarrow 0$
2. Tant que non condition d'arrêt Faire
   Pour chaque fourmi k=1..nbAnts Faire
   Choisir un sommet $i \in V$ aléatoirement
   $S_k \leftarrow \{i\}$
   Tant que $|S_k| \leq |V| - 1$ Faire
   Choisir $j \in V - S_k$ selon la probabilité $p_{ij}^k(1)$
   $S_k \leftarrow S_k \cup \{j\}$
   $i \leftarrow j$
   Fin Tant que
   Fin Pour
   Pour chaque arrête $(i, j) \in E$ Faire
   Mettre à jour la trace $\tau_{ij}(t)$ avec la formule (2)

6.1.4.1 Extensions de Ant System

Malgré ses résultats encourageants, AS n'était pas compétitif avec les algorithmes de l'état de l'art du PVC. Des recherches sont alors entreprises pour l'étendre et essayer d'améliorer ses performances.

Rank-based Ant system (AS-rank)

C'est une extension proposée par Bullnheimer, Hartl, and Strauss [108]. Cet algorithme trie les fourmis selon les longueurs de tours générés. Après chaque phase de construction de tours, seulement les *w* meilleures fourmis sont autorisées à la mise à jour de phéromone. Ainsi, la $r^{ième}$ fourmi contribue à la mise à jour de phéromone avec un poids donné par le max {0, w-r} tandis que la fourmi correspondant au meilleur tour global renforce la trace de phéromone avec un poids *w*. L'équation de mise à jour devient alors :

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + w\Delta\tau_{ij}^{gb} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r}$$

Ant-Q

Dans Ant-Q [108], la règle de mise à jour locale est inspirée du "Q-learning" : un algorithme d'apprentissage par renforcement. Cependant, aucune amélioration par rapport à l'algorithme AS n'a pu être démontrée. Cet algorithme n'est d'ailleurs qu'une pré-version du "Ant Colony System".

Ant Colony System

L'algorithme "Ant Colony System" (ACS) [131] a été introduit par Dorigo et Gambardella pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles. L'ACS introduit une règle qui dépend d'un paramètre q_0 $(0 \le q_0 \le 1)$, qui définit une balance diversification/intensification. La fourmi *k* sur une ville *i* choisit la ville *j* de destination selon:

$$J = \begin{cases} \arg\max_{u \in J_i^k} (\tau_{ij}(t)\eta_{ij}^{\beta}) & \text{si } q \le q0 \\ J & \text{si } q > q0 \end{cases}$$

Où *j* est une ville tirée au hasard dans j_i^k avec la probabilité suivante :

$$p_j = \frac{\tau_{ij}(t).\eta_{ij}^{\beta}}{\sum_{l \in J_i^k} \tau_{ij}(t).\eta_{ij}^{\beta}}$$

La mise à jour de phéromone est réalisée suivant une règle globale et une autre locale :

Règle de mise à jour de phéromone globale

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \rho.\Delta\tau_{ij}(t)$$

où seules les arêtes (i,j) du meilleure tour T_{best} sont modifiées $\Delta\tau_{ij} = 1/L_{\min}$ où $L_{\min}$ est la longueur de T_{best}.

Règle de mise à jour de phéromone locale

Chaque fourmi dépose sur les arcs (i,j) qu'elle a visité une quantité $\rho\tau 0$ de phéromone où $\tau 0$ est la valeur initiale de la piste, alors qu'une quantité $(1-\rho)\tau_{ij}$ s'évapore :

$$\tau'_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \rho.\tau 0$$

où $\tau 0$ est une paramètre (par exemple $\frac{1}{N.L}$ où L est l'estimation d'un chemin de bonne qualité).

MAX −MIN Ant Sytem (MMAS)

Stützle et Hoos ont introduit l'algorithme MMAS [122] qui, pour améliorer les performances de ACO, combine une exploitation améliorée des meilleures solutions trouvées durant la recherche avec un mécanisme efficace pour éviter une stagnation prématurée de la recherche. MMAS diffère en trois aspects clé de AS :

- Pour exploiter les meilleures solutions trouvées durant une itération ou durant l'exécution de l'algorithme, après chaque itération, une seule fourmi ajoute de la phéromone.
  Cette fourmi peut être celle qui a trouvé la meilleure solution depuis le début de l'exécution, ou bien celle durant le dernier cycle.
- Pour éviter une stagnation de la recherche, les valeurs possibles de la trace de phéromone sur chaque composant de solution sont limitées à l'intervalle $[\tau_{\min}, \tau_{\max}]$.
- De plus, les traces de phéromone sont initialisées à $\tau_{\max}$ pour assurer une exploration élevée de l'espace des solutions au début de l'algorithme.

6.1.5 La métaheuristique d'optimisation par colonie de fourmis

En utilisant la métaheuristique ACO, l'une des tâches principales est d'encoder le problème à résoudre sous la forme d'un problème de recherche d'un chemin optimal dans un graphe pondéré, appelé graphe de construction, où la faisabilité est définie en respectant un ensemble de contraintes. Les fourmis artificielles peuvent être caractérisées comme des procédures de construction stochastiques qui construisent des solutions en se déplaçant dans le graphe de construction.
Dans cette section, nous définissons d'abord une représentation générique du problème que les fourmis exploitent pour construire des solutions. Ensuite, nous détaillons le comportement des fourmis durant la construction de solution et finalement nous définissons la métaheuristique d'optimisation par colonie de fourmis.

6.1.5.1 Représentation du problème

Considérons le problème de minimisation (idem pour la maximisation) (S, f, Ω) où S est l'ensemble des solutions candidates, f la fonction objectif qui associe à chaque solution candidate $s \in S$ une valeur de la fonction objectif *f(s)* et Ω est l'ensemble des contraintes. L'objectif est de trouver une solution optimale globale $s_{opt} \in S$ qui est une solution minimisant f et qui satisfait les contraintes Ω. Les différents états du problème sont caractérisés comme une séquence de composants.
Les fourmis artificielles construisent les solutions en se déplaçant sur le graphe construit G(C, L) tel que les sommets sont les composants **C** et **L** est l'ensemble qui connecte les composants de C (un élément de L est une connexion). Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

6.1.5.2 Comportement des fourmis

Les fourmis peuvent être caractérisées comme des procédures de construction stochastiques qui construisent des solutions en se déplaçant sur le graphe construit G(C, L). Les fourmis ne se déplacent pas arbitrairement, mais elles suivent une politique de construction qui est une fonction des contraintes du problème.
Les traces de phéromone τ peuvent être associées aux composants ou aux connexions du graphe (τ_i pour un composant, τ_{ij} pour une connexion) créant une mémoire à long terme sur la totalité du processus de recherche de la fourmi, qui change dynamiquement durant la résolution afin de refléter l'expérience de recherche acquise par les agents, et une valeur heuristique η (η_i , η_{ij} respectivement) représentant une information sur l'instance du problème ou une information sur le temps d'exécution fourni par une source autre que les fourmis. Dans la plupart des cas, η est le coût ou une estimation du coût de l'extension de l'état courant.

6.1.5.3 La métaheuristique ACO

Dans les algorithmes ACO [135,136], les fourmis se déplacent en appliquant une politique de décision stochastique locale qui se base sur l'utilisation des traces de phéromone et d'une information heuristique spécifique au problème traité. Des coefficients α et β permettent de contrôler l'importance relative des deux éléments. Chaque fourmi *k* possède une forme de mémoire, $tabou_{k,}$ afin d'obliger celle-ci à former une solution admissible.

En se déplaçant, les fourmis construisent par incréments des solutions au problème d'optimisation. Une fois qu'une fourmi a construit une solution, ou lorsque la solution est en cours de construction, la fourmi évalue la solution (partielle) et dépose la phéromone dans le composant ou la connexion qu'elle a utilisé. Cette information de phéromone dirigera la recherche des futures fourmis.
L'intensité de ces traces décroît dans le temps par un facteur constant appelé coefficient d'évaporation. D'un point de vue pratique, l'évaporation de phéromone est nécessaire pour éviter une convergence prématurée de l'algorithme vers une région sous-optimale. Ce processus implémente une forme utile d'oubli favorisant l'exploration de nouvelles aires de recherche.

6.1.6 Applications de Ant System

Les algorithmes ACO ont été appliqués à plusieurs problèmes combinatoires NP-difficiles. L'application de ACO au PVC a été déjà présentée dans la section précédente. Dans cette section, nous discutons des applications à d'autres problèmes d'optimisation combinatoire.

6.1.6.1 Problème de Tournée de Véhicules (VRP)

Parmi les applications les plus intéressantes de la métaheuristique ACO au VRP, nous citons l'algorithme AS-VRP conçu par Bullnheimer, Harlt et Strauss [108] qui est basée sur l'algorithme AS_{rank} .

Gambardella, Taillard et Agazzi [111] ont traité le VRP en adaptant l'approche ACS pour définir MACS-VRPTW, et en considérant l'extension de la fenêtre temps au VRP qui introduit un intervalle de temps dans lequel un client doit être servi.

6.1.6.2 Le Problème de k-Partitionnement de Graphes

Le problème de k-partitionnement de graphes (k-PPG) consiste à partitionner les nœuds d'un graphe donné en *k* partitions de façon à minimiser les coûts des arcs inter-partitions tout en essayant d'assurer l'équilibre des poids des partitions.
Ce problème a été traité par la métaheuristique ACO par Alaya, Sammoud, Hammami et Ghédira [137]. Dans l'algorithme proposé, le graphe de construction décrivant ce problème est un graphe complet dont les noeuds sont des couples $\langle X, P \rangle$ avec X est un noeud du graphe à partitionner et P est le numéro de partition à laquelle X peut appartenir. Ainsi, un couple $\langle X, P \rangle$ choisi par une fourmi signifie que le noeud X du graphe est affecté à la partition P, et

par conséquent tout couple $\langle X, Q \rangle$ tel que $Q \neq P$ devient interdit (n'appartient plus au voisinage faisable) puisque qu'un noeud ne peut pas appartenir à plus qu'une partition à la fois.

Au cours de chaque cycle, chaque fourmi produit un partitionnement en se déplaçant sur le graphe de construction : à partir du noeud courant $\langle X, P \rangle$, elle choisit de se délacer vers le nœud $\langle Y, Q \rangle$ selon une probabilité qui dépend :

- du coût des arcs inter-partitions apporté par ce nouveau composant (l'information heuristique),
- des traces de phéromone.

La quantité de phéromone à déposer est inversement proportionnelle au coût de la solution générée.

6.1.6.3 Le problème du sac à dos multidimensionnel

Le MKP présente l'un des problèmes les plus étudier et sur lesquels nous travaillons. Dans cette section, nous présentons les travaux proposés dans la littérature pour résoudre ce problème avec ACO.

Une application de ACO au MKP définit le graphe de construction comme étant un graphe complet composé des objets du MKP (qui représentent les noeuds). La mise à jour de phéromone se fait sur les composants des solutions (c'est à dire les noeuds du graphe). Les traces de phéromone reflète donc la désirabilité de prendre un objet qui a fait partie de plusieurs solutions. La quantité de phéromone à déposer est égale au profit de la solution construite. L'information heuristique est calculée de façon dynamique en fonction des capacités des ressources (les contraintes) et du profit de l'objet à prendre.

Une autre application d'ACO au MKP a été présentée dans [126]. Dans cette application, les traces de phéromone sont déposées sur les arêtes visitées. Après que toutes les fourmis aient complété leurs tours, les fourmis déposent de la phéromone sur les chemins qu'elles viennent de parcourir. De même dans cette application la quantité de phéromone à déposer est égale au profit de la solution construite. Ensuite, la meilleure fourmi du cycle renforce les traces de phéromone sur les composants de la meilleure solution. De plus lorsque certains mouvements ne sont pas utilisés dans la solution courante, un renforcement de phéromone additionnelle est utilisé pour ces composants afin de les favoriser dans les itérations futures. L'information heuristique est calculée de façon statique.

6.1.7 ACO et la recherche locale

Dans plusieurs applications à des problèmes d'optimisation combinatoire NP-difficiles, les algorithmes ACO réalisent de meilleures performances lorsqu'ils sont combinés avec des algorithmes de recherche locale [131]. Ces algorithmes optimisent localement les solutions des fourmis et ces solutions optimisées localement sont utilisées par la suite dans la mise à jour de phéromone. L'utilisation de la recherche locale dans les algorithmes ACO peut être très intéressante comme les deux approches sont complémentaires. En effet, la combinaison peut améliorer largement la qualité des solutions produites par les fourmis.
D'un autre côté, générer des solutions initiales pour les algorithmes de recherche locale n'est pas une tâche facile. Dans les algorithmes de recherche locale, où les solutions initiales sont générées aléatoirement, la qualité des solutions peut être médiocre. En combinant la recherche locale avec l'ACO, les fourmis génèrent stochastiquement, en exploitant les traces de phéromone, des solutions initiales prometteuses pour la recherche locale.

6.1.8 Conclusion

ACO est une méthode stochastique qui requiert de plus en plus l'attention de la communauté scientifique. Cette métaheuristique a prouvé sa performance pour plusieurs problèmes d'optimisation combinatoire NP-difficiles.
L'utilisation des traces de phéromone permet d'exploiter l'expérience de recherche acquise par les fourmis et ainsi renforcer l'apprentissage pour la construction de solutions dans les itérations futures de l'algorithme. En même temps, l'information heuristique peut guider les fourmis vers les zones prometteuses de l'espace de recherche.
Un autre avantage de ACO est que son domaine d'application est vaste. En principe, ACO peut être appliquée à n'importe quel problème d'optimisation discret pour lequel un mécanisme de construction de solution peut être conçu.
Toutefois, ACO a ses propres inconvénients qui résident, principalement, dans les problèmes de paramétrage. En effet, ACO nécessite une instanciation des différents paramètres qui dépend du problème. Généralement, les paramètres correspondant aux meilleures solutions prouvées expérimentalement sont retenus.

6.2. Un algorithme ACO générique pour la résolution de PMO

6.2.1 Introduction

Pour résoudre un problème d'optimisation multi-objectif (PMO) avec ACO, plusieurs points sont à définir pour lesquels différents choix sont possibles. Nous avons étudié dans le chapitre 4 plusieurs algorithmes MOACO qui ont été proposés dans la littérature pour résoudre des PMO. Nous avons établi une taxonomie de ces algorithmes suivant différents critères de classement.

Dans ce chapitre, nous proposons une nouvelle variante [138] d'un algorithme ACO générique, appelé m-ACO [139], pour la résolution de PMO, et plus particulièrement les problèmes de sélection de sous-ensembles.

Cet algorithme générique nous permet de mettre dans un même cadre différentes approches que nous pouvons tester et comparer sans se soucier des détails d'implémentation ni du schéma ACO considéré. Nous instancions l'algorithme m-ACO par quatre variantes et nous proposons une nouvelle variante. Il est à noter que nous considérons, dans cet algorithme, les problèmes de maximisation (dans le cas de minimisation il suffit de multiplier la fonction objectif par -1).

6.2.2 L'algorithme générique m-ACO

Dans cette section, nous présentons un algorithme ACO générique pour des problèmes multi-objectif. Cet algorithme sera instancier et les variantes seront décrites par la suite.

L'algorithme générique, appelé m-ACO, est implicitement paramétré par le PMO (X,D,C, F) à résoudre. Nous considérons que les fourmis construisent les solutions dans un graphe G = (V,E) dont la définition dépend du problème à résoudre et que les traces de phéromone sont associées aux sommets ou arêtes de ce graphe. Nous assumons aussi que l'information heuristique est définie pour chaque fonction objectif $f_i \in F$.

m-ACO est aussi paramétré par le nombre de colonies de fourmis #Col et le nombre de structures de phéromone considérées $\#\tau$. La figure 4.1 décrit l'algorithme générique de m-ACO(#Col, $\#\tau$). L'algorithme suit le schéma de MAX−MIN Ant System. Les traces de phéromone sont initialisées d'abord à une borne supérieure donnée $\tau_{\max}$.

Par la suite, à chaque cycle chaque fourmi construit une solution et les traces de phéromone sont mises à jour. Pour éviter une convergence prématurée, les traces de phéromone sont

bornées avec deux bornes τ_{min} et τ_{max} telles que $0 < \tau_{min} < \tau_{max}$. L'algorithme s'arrête lorsqu'un nombre maximum de cycles est atteint.

6.2.2.1 Construction de solution

La figure 6.2 décrit l'algorithme utilisé par les fourmis pour construire des solutions dans le graphe de construction G = (V,E) dont la définition dépend du problème (X,D,C, F) à résoudre. A chaque itération, un sommet de G est choisi parmi un ensemble de sommets candidats Cand ; il est ajouté à la solution S et l'ensemble des sommets candidats est mis à jour en supprimant les sommets qui violent des contraintes de C. Le sommet v_i à ajouter à la solution S par les fourmis de la colonie c est choisi relativement à la probabilité $p_S^c(v_i)$ définie comme suit :

$$p_S^c(v_i) = \frac{[\tau_S^c(v_i)]^\alpha . [\eta_S^c(v_i)]^\beta}{\sum_{v_j \in Cand} [\tau_S^c(v_j)]^\alpha . [\eta_S^c(v_j)]^\beta}$$

Où $\tau_S^c(v_i)$ et $\eta_S^c(v_i)$ sont respectivement les facteurs phéromone et heuristique du sommet candidat v_i, et α et β sont deux paramètres qui déterminent leur importance relative. La définition du facteur phéromone dépend des paramètres #Col et $\#\tau$, il va être détaillé par la suite. La définition du facteur heuristique dépend du problème à résoudre. Il doit être défini lors de l'application de l'algorithme à un problème spécifique.

Algorithme m-ACO$(\#Col, \#\tau)$ **:**

Initialiser les traces de pheromone a τ_{max}

repeter

pour chaque colonie c **dans** $1..\#Col$

pour chaque fourmi k **dans** $1..nbf$

construire une solution

pour i **dans** $1..\#\tau$

mettre a jour la $i^{ème}$ structure de traces de pheromone

si une trace est inferieure a τ_{min} alors la mettre a τ_{min}

si une trace est superieure a τ_{max} alors la mettre a τ_{max}

jusqu'a un nombre maximal de cycles atteint

Fig. 6.2 - Algorithme ACO générique pour POM

Construction d'une solution S **:**

$S \leftarrow \emptyset$

$Cand \leftarrow V$

tant que $Cand \neq \emptyset$ **faire**

choisir $v_i \in Cand$ avec probabilite $p_S(v_i)$

ajouter v_i a S

supprimer de $Cand$ les sommets qui violent des contraintes

fin tq

Fig. 6.3 - Construction de solution

6.2.2.2 Mise à jour de phéromone

Une fois que toutes les fourmis aient construites leurs solutions, les traces de phéromone sont mises à jour : d'abord, les traces de phéromone sont réduites par un facteur constant pour simuler l'évaporation ; par la suite, des traces de phéromone sont déposées sur les meilleures

solutions. Plus précisément, la quantité $\tau^i(c)$ de la $i^{ème}$ structure de phéromone déposée sur le composant c est mise à jour comme suit :

$$\tau^i(c) \leftarrow (1-p) * \tau^i(c) + \Delta\tau^i(c)$$

où p est le facteur évaporation, tel que $0 \leq p \leq 1$, et $\Delta\tau^i(c)$ est la quantité de phéromone déposée sur le composant c. La définition de cette quantité dépend des paramètres #Col et #τ ; elle va être détaillée par la suite.

6.2.3 Instanciation de l'algorithme générique m-ACO

Nous décrivons, dans cette section, quatre variantes de cet algorithme générique et nous proposons une nouvelle variante utilisant une nouvelle stratégie phéromonale qui considèrent différentes valeurs pour les paramètres nombres de colonies de fourmis #Col et structures de phéromone $\#\tau$. Ces variantes diffèrent aussi dans la phase de mise à jour de phéromone. Parmi ces variantes, nous retrouvons des approches ACO qui existaient déjà dans la littérature et d'autres nouvelles approches proposées. Etant donné qu'il est difficile d'implémenter tous les algorithmes MOACO proposés dans la littérature, nous avons repris l'idée de base des meilleurs algorithmes retournés par la comparaison de plusieurs MOACO réalisé dans [119] sur le problème du voyageur de commerce multi-objectif.

6.2.3.1 Variante 1 : m-ACO1(m+1,m)

Pour cette variante, le nombre de colonies #Col est mis à $m + 1$ et le nombre de structures de phéromone est mis à m, où $m = |F|$ est le nombre d'objectifs à optimiser : chaque colonie considère un objectif différent, et utilise sa propre structure de phéromone ; une autre colonie multi-objectif est ajoutée, qui optimise tous les objectifs.

Définition des facteurs phéromone. Le facteur phéromone $\tau_S^i(v_j)$ considéré par la i^{ème} colonie uni-objectif, qui optimise la i^{ème} fonction objectif f_i, est défini relativement à la i^{ème} structure phéromone ; en fonction de l'application considérée, il peut être défini comme étant la quantité de phéromone déposée sur le sommet v_j ou comme étant la quantité de phéromone déposée sur les arêtes entre v_j et quelques sommets dans la solution partielle S . Le facteur phéromone $\tau_S^{m+1}(v_j)$ considéré par la colonie multi-objectif est le facteur phéromone $\tau_S^r(v_j)$ de la r^{ème} colonie uni-objectif, où r est une colonie choisie aléatoirement.

Ainsi cette colonie considère, à chaque étape de construction, un objectif à optimiser choisi aléatoirement.

Mise à jour de phéromone. Pour chaque colonie uni-objectif, la phéromone est déposée sur les composants de la meilleure solution trouvée par la i^{ème} colonie durant le cycle courant, où la qualité de solutions est évaluée relativement au i^{ème} objectif *fi* seulement. Ainsi, soit S^i la meilleure solution du cycle courant de la i^{ème} colonie qui optimise la fonction *fi*, et soit S^i_{meil} la meilleure solution de la i^{ème} colonie parmi toutes les solutions construites par les fourmis de la i^{ème} colonie depuis le début de l'exécution de l'algorithme. La quantité de phéromone déposée sur un composant de solution *c* pour la i^{ème} structure de phéromone est définie comme suit :

$$\Delta\tau^i(c) = \frac{1}{(1 + f_i(S^i) - f_i(S^i_{best}))} \qquad \text{Si c est un composant de } S^i$$

$$\Delta\tau^i(c) = 0 \qquad \text{Sinon.}$$

La colonie multi-objectif maintient un ensemble de solutions : une meilleure solution pour chaque objectif. Elle dépose de la phéromone sur chaque structure de phéromone relativement à l'objectif correspondant avec la même formule définie pour les autres colonies.

6.2.3.2 Variante 2 : m-ACO2(m+1,m)

Cette deuxième variante est très similaire à la première, et considère m+1 colonies et m structures de phéromone : une colonie uni-objectif est associée à chaque objectif, et le comportement de ces colonies est défini comme dans la première variante ; il y a aussi une autre colonie supplémentaire multi-objectif qui optimise tous les objectifs. La seule différence entre la variante 1 et 2 réside dans la manière avec laquelle cette colonie multi-objectif exploite les structures de phéromone des autres colonies pour construire des solutions. Pour cette colonie multi-objectif, le facteur phéromone $\tau_S^{m+1}(v_j)$ est égal à la somme de chaque facteur phéromone $\tau_S^c(v_j)$ de chaque colonie $c \in \{1,...,m\}$, agrégeant ainsi toutes les informations phéromone dans une seule valeur.

Cette variante reprend l'idée de base de l'algorithme CompetAnts [113].

Cependant CompetAnts possède quelques spécificités. D'abord, le nombre de fourmis dans les colonies n'est pas fixe. De plus, les auteurs ne définissent pas une colonie *m + 1* qui

optimisent tous les objectifs mais un certain nombre de fourmis dans chaque colonie qu'ils appellent "spies".

6.2.3.3 Variant 3 : m-ACO3(1,1)

Dans cette instanciation de m-ACO, il y a une seule colonie de fourmis et une seule structure de phéromone, i.e., #Col = 1 et $\#\tau = 1$.

Définition des facteurs phéromone. Le facteur phéromone $\tau_S^1(v_j)$ considéré par les fourmis de l'unique colonie est défini relativement à l'unique structure de phéromone ; en fonction de l'application considérée, il peut être défini comme étant la quantité de phéromone déposée sur le sommet v_j ou comme étant la quantité de phéromone déposée sur les arêtes entre v_j et quelques sommets dans la solution partielle S.

Mise à jour de phéromone. Une fois la colonie ait construit un ensemble de solutions, chaque solution non-dominée (appartenant à l'ensemble Pareto) est récompensée.
Soit S_P l'ensemble des solutions non-dominées. La quantité de phéromone déposée sur un composant de solution c est défini comme suit :

$\tau^1(c) = 1$ Si c est un composant d'une solution de S_P

$\tau^1(c) = 0$ Sinon.

Chaque composant appartenant à au moins une solution de l'ensemble Pareto reçoit la même quantité de phéromone. En effet, ces solutions appartiennent à des solutions non comparables.
Cette variante reprend l'idée de base de l'algorithme MACS [112].
La seule différence est dans la définition de la quantité à déposer lors de la mise à jour de phéromone. Dans MACS, le dépôt de phéromone est aussi autorisé aux solutions non dominées mais avec une quantité obtenue en agrégeant les fonctions objectifs et non pas une quantité fixe.

6.2.3.4 Variante 4 : m-ACO4(1,m)

Dans cette variante, il y a une seule colonie mais m structures de phéromone, i.e., #Col = 1 et $\#\tau$ = m.

Facteur phéromone. A chaque étape de construction de solution, les fourmis choisissent aléatoirement un objectif $r \in \{1,...,m\}$ à optimiser. Le facteur phéromone $\tau_S^1(v_j)$ est ainsi défini par le facteur phéromone associé l'objectif choisi aléatoirement r.

Mise à jour de phéromone. Une fois la colonie ait construit un ensemble de solutions, les m meilleures solutions relativement aux m objectifs sont utilisées pour récompenser les m structures de phéromone. Soit S^i la solution de la colonie qui minimise le i^{ème} objectif f_i pour le cycle courant, et soit S^i_{meil} la solution qui minimise *fi* parmi toutes les solutions construites par les fourmis depuis le début de l'exécution. La quantité de phéromone déposée sur un composant de solution *c* pour la i^{ème} structure de phéromone est définie comme suit:

$$\Delta\tau^i(c) = \frac{1}{(1 + f_i(S^i) - f_i(S^i_{best}))} \quad \text{Si c est un composant de } S^i$$

$$\Delta\tau^i(c) = 0 \quad \text{Sinon.}$$

Chapitre 7 : Contribution Principale et Résultats d'Expérimentations :

7.1 Introduction :

Dans cette section on propose une nouvelle variante de l'algorithme m-ACO qui présente la contribution principale de notre travail [140]. Cette variante utilise le principe des méthodes Pareto. Ces méthodes utilisent directement la notion de dominance au sens de Pareto dans la sélection des solutions générées (voir chapitre 1 et 2). Cette idée a été initialement introduite par Goldberg [141] pour résoudre les problèmes proposés par Schaffer [142].

Dans ce type de méthode, deux phases importantes sont à considérer: la phase de recherche de l'ensemble des solutions Pareto optimales, que nous appellerons de façon abusive, résolution du problème d'optimisation et la phase de choix parmi ces solutions, qui relève de l'aide à la décision. Cette deuxième phase ne sera pas traitée ici, car on ne cherche pas la meilleure solution parmi les solutions de cet ensemble mais on cherche à avoir un ensemble (le plus complet possible) de solutions Pareto, afin qu'on assure la qualité des solutions (ce qui nous intéresse le plus) pour mettre à jour les traces de phéromone. En effet, tous les algorithmes MOACO utilisant l'approche Pareto, se trouvent aussi parmi les meilleurs résultats comparé avec plusieurs algorithmes MOACO de l'état de l'art dans [119] pour le problème du voyageur de commerce. Ainsi, nous avons constaté que pour les algorithmes MOACO, tout comme les algorithmes évolutionnaires, les approches Pareto semblent trouver les meilleurs résultats d'où vient l'idée d'autoriser les solutions non dominées de l'ensemble Pareto à déposer sur les structures de phéromones .

7.2 Nouvelle Variante : NV-m-ACO(1,m)

Cette nouvelle variante utilise une nouvelle stratégie phéromonale pour la mise à jour de phéromone.
Cette nouvelle variante est très similaire à la précédente (m-ACO4(1,m))et utilise une seule colonie et m structures de phéromone. Le comportement des fourmis de la colonie est défini

comme dans la variante précédente : à chaque étape transition, les fourmis choisissent aléatoirement un objectif $r \in \{1,...,m\}$ à optimiser**.**

Lors de la mise à jour de phéromone, les m meilleures solutions relativement aux m objectifs sont utilisées pour récompenser les m structures de phéromone exactement comme dans la variante précédente. De plus, toutes les solutions non-dominées de l'ensemble Pareto sont aussi autorisées à déposer sur les m structures de phéromone.

Soit S_P l'ensemble des solutions non-dominées. La quantité de phéromone déposée sur un composant de solution c pour la i^{ème} structure de phéromone est défini comme suit :

$\tau^i(c) = 1$ Si *c* est un composant d'une solution de S_P

$\tau^i(c) = 0$ Sinon.

Begin NV-m-ACO(1,m)
Input: α , β , ρ , $\tau_{\min}$, $\tau_{\max}$, *m ,nb-ants, nb-cyc,* τ
lPareto –list of non dominated solutions that will be updated with S.
Output: S
Initialize all pheromone trails to $\tau_{\max}$
lPareto ← ∅
Do while maximal number of cycle is not reached
for each colony in 1.. C
for each ant in 1..nbAnts
% construct a solution
$S \leftarrow \varnothing$
Cand ← *V*
While Cand ≠ ∅ *do*
choose $v_i \in$ *Cand with probability* $p_S(vi)$
add at the end of
remove from Cand vertices that violate constraints.
End while.
lPareto =non dominated solutions of (lPareto ∈ **{S});**
% end construct a solution
for i in 1.. τ
Update the ith pheromone structure trails By S_{best} and lPareto. (Applying the min-max principle)
End.

Fig. 7.1 – Nouvelle variante NV-m-ACO(1,m) .

Dans le chapitre suivant nous montrons comment la nouvelle variante de l'algorithme générique m-ACO proposé dans ce chapitre peut être appliqué au problème du sac à dos multi-objectif (MOKP). Nous testons et comparons les variantes décrites dans ce chapitre sur des instances benchmark du MOKP.

7.3 Discussion :

Nous avons proposé dans le début de ce chapitre une nouvelle variante d'un algorithme générique basé sur l'optimisation par colonie de fourmis pour la résolution des problèmes d'optimisation multi-objectifs. Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de traces de phéromone. Nous considérons que le nombre de colonies de fourmis peut être différent du nombre de structures de phéromone, alors que généralement dans les algorithmes MOACO ces deux paramètres sont considérés comme étant un seul. Nous avons présenté cinq variantes de cet algorithme. Parmi ces variantes, les quatre première reprennent le schéma général d'autres algorithmes MOACO présentés dans la littérature la dernière présente une nouvelle approche.
La variante m-ACO1(m+1,m) utilise une colonie multi-objectif qui considère à chaque étape de construction un objectif choisi aléatoirement. Les deux variantes m-ACO4(1,m) et NV-m-ACO(1,m) emploient une seule colonie avec plusieurs structures phéromone. De plus, les fourmis de la colonie considèrent à chaque étape de construction un objectif aléatoirement. Comme nous l'avons souligné au début de ce chapitre, cet algorithme générique nous permet de mettre dans un même cadre différentes approches que nous pouvons tester et comparer sans se soucier des détails d'implémentation ni du schéma ACO considéré. Dans le chapitre suivant, nous appliquons ces différentes variantes sur le problème du sac à dos multi-objectif.

7.4 Expérimentations et résultats (Application du m-ACO au problème du sac à dos Multidimensionnel multi-objectif)

7.4.1 Introduction

Nous montrons dans ce chapitre comment l'algorithme générique m-ACO proposé dans le chapitre précédent peut être appliqué au problème du sac à dos multi-objectif (MOKP). Nous testons et comparons les variantes décrites dans le chapitre précédent sur des instances benchmark du MOKP. Nous comparons aussi notre approche avec des algorithmes évolutionnaires de l'état de l'art.

7.4.2 Le problème du sac à dos multidimensionnel multi-objectif

Ce problème est parmi les problèmes les plus étudiés dans la communauté multi-objectif. L'objectif de ce problème est de maximiser un vecteur de fonction profit tout en satisfaisant un ensemble de contraintes de capacité du sac à dos. Plus formellement, un MOKP est défini comme suit :

$$\text{Maximiser } f_k = \sum_{j=1}^{n} p_j^k x_j, \forall k \in 1..m$$

$$\text{s.c } \sum_{j=1}^{n} w_j^i x_j \leq b_i, \forall i \in 1..q$$

$$x_j \in \{0,1\}, \forall j \in 1..n$$

Où

m est le nombre de fonctions objectif,

n est le nombre d'objets,

x_j est la variable de décision associée à l'objet o_j ,

q est le nombre de contraintes de ressource,

w_j^i est la quantité de la ressource *i* consommé par l'objet o_j ,

b_i est la quantité totale disponible de la ressource *i*,

p_j^k est le profit associé à l'objet o_j relativement l'objectif *k*.

7.4.3 Choix de la stratégie phéromonale

Nous avons étudié dans le chapitre 3 différentes stratégies pour la définition de la structure de phéromone pour le problème du sac à dos multidimensionnel uni-objectif. Nous rappelons brièvement dans ce qui suit ces différentes stratégies. En effet, pour résoudre un problème de type sac à dos, un point clé est de décider quels composants des solutions construites doivent être récompensés, et comment exploiter ces récompenses lors de la construction de nouvelles solutions. Etant donnée une solution d'un problème sac à dos $S = \{o_1 ..., o_k\}$, nous pouvons considérer trois manières différentes de déposer les traces de phéromone :

- Une première possibilité est de déposer la phéromone sur chaque objet sélectionné dans S .
- Une deuxième possibilité est de déposer la phéromone sur chaque couple $\{o_i, o_{i+1}\}$ d'objets sélectionné successivement dans S .
- Une troisième possibilité est de déposer la phéromone sur chaque paire $\{o_i, o_j\}$ d'objets différents de S .

Nous avons trouvé que la première et troisième stratégie trouvent des résultats nettement meilleurs que la deuxième. Ceci est expliqué par le fait que pour les problèmes de type sac à dos l'ordre dans lequel les objets sont sélectionnés n'est pas significatif. Cependant, la différence entre les deux premières stratégies n'est pas aussi importante même si on note un léger avantage pour la première stratégie sur la qualité des solutions. Cependant la deuxième stratégie trouve des résultats très proches avec un temps de calcul inférieur que la première, puisque cette dernière est de complexité quadratique alors que la deuxième est de complexité linéaire.

Etant donné la difficulté simultanée du cadre multi-objectif et de la complexité de la deuxième stratégie, nous avons choisi d'utiliser, pour le problème du sac à dos multidimensionnel multi-objectif, la première possibilité puisqu'elle offre un bon compromis entre la qualité des solutions et le temps d'exécution.

7.4.4 Définition du facteur heuristique

Nous définissons le facteur heuristique pour le problème du sac à dos multidimensionnel multi-objectif comme étant une agrégation du facteur heuristique nous avons utilisé pour le problème du sac à dos multidimensionnel uni-objectif défini dans le chapitre 5. Ainsi, le facteur heuristique $\eta_S(o_j)$ utilisé dans la probabilité de transition pour le choix d'un objet candidat o_j est défini comme suit : soit $d_S(i) = b_i - \sum_{g \in S} r_{ig}$ la quantité restante de la ressource i lorsque la fourmi a construit la solution S ; nous définissons le ratio $h_S(o_j) = \sum_{i=1}^{q} \frac{w_j^i}{d_S(i)}$ qui présente la dureté de l'objet o_j par rapport à toutes les contraintes $i \in 1..q$ et relativement à la solution construite S , de sorte que plus ce ratio est faible plus l'objet est intéressant.

Nous intégrons alors le profit p_j^k de l'objet o_j par rapport à l'objectif *k* pour obtenir un ratio profit/ressource correspondant à l'information heuristique relative à l'objectif *k*,

$$\eta_S^k(o_j) = \frac{p_j^k}{h_S(o_j)}.$$

Nous définissons alors le facteur heuristique par une agrégation des informations heuristiques relatives à tous les objectifs,

$$\eta_S(o_j) = \sum_{k=1}^{m} \eta_S^k(o_j).$$

7.4.5 Expérimentations et résultats

Nous présentons dans cette section les résultats des variantes de l'algorithme m-ACO, i.e., m-ACO1(m+1,m), m-ACO2(m+1,m), m-ACO3(1, 1), m-ACO4(1,m) et NV-m-ACO(1,m). Nous comparons aussi ces résultats avec plusieurs algorithmes évolutionnaires de l'état de l'art : SPEA [143], NSGA [144], HLGA [145], NPGA [146], VEGA [147], SPEA2 [148] et NSGAII [149]. Ces algorithmes évolutionnaires ont été décrits dans le chapitre 2.

7.4.5.1 Ensemble de tests

Les expérimentations sont effectuées sur les 9 instances définies dans [150]. Ces benchmarks ont 2, 3 et 4 objectifs combinés avec 250, 500 et 750 objets. De plus, pour chaque instance, le nombre de contraintes est égal au nombre d'objectifs. Ces instances ont été générées de manière aléatoire avec des poids et des profits décorélés, et les capacités de chaque contrainte sont réglées à environ la moitié de la somme des poids de la contrainte considérée. Pour cela le nombre d'objets présents attendu dans les solutions optimales est d'environ la moitié du nombre total d'objets du problème. Nous allons noter ces instances dans ce qui suit *n.m* avec *n* étant le nombre d'objets et *m* le nombre d'objectifs.
Les instances considérées sont accessibles sur le site http://www.tik.ee.ethz.ch/pisa/

7.4.5.2 Conditions d'expérimentation et paramétrage

Pour toutes les variantes de m-ACO(#Col,#τ), nous avons considéré les mêmes conditions d'expérimentations et les mêmes valeurs pour les paramètres de l'algorithme. Nous avons mis α , le coefficient de pondération du facteur phéromone, à 1, et ρ , le ratio d'évaporation, à 0.01. Nous nous sommes basés pour le choix de ces paramètres sur l'étude que nous avons présentée au chapitre 5 sur l'influence de ces paramètres sur la résolution pour le cas du problème du sac à dos multidimensionnel uni-objectif. En effet, nous avons montré dans ce chapitre que la recherche d'un bon compromis entre intensification et diversification de la recherche peut être influencé par ces paramètres.
Nous avons trouvé que, avec ces valeurs de α et ρ qui favorisent l'exploration, les fourmis trouvent de meilleures solutions en fin d'exécution, mais elles ont besoin de plus de cycles pour converger sur ces solutions.

Pour le choix des valeurs des autres paramètres nous avons mené une série d'expérimentations sur quelques instances du MOKP. Nous avons mis β, le coefficient de pondération du facteur heuristique à 4, le nombre de cycles à 3000 et les bornes de phéromone $\tau_{\min}$ et $\tau_{\max}$ à 0.01 et 6. Nous avons mis le nombre de fourmis pour chaque colonie employée à $\frac{100}{\#Col}$, ainsi nous obtenons le même nombre de solutions générées pour toutes les variantes.
Chaque variante a été testée dix fois pour chaque instance reportée. Donc nous présentons par la suite pour les métriques considérées les résultats calculés de dix exécutions de chaque algorithme.

7.4.5.3 Mesures de performance considérées

Pour comparer les performances des différents algorithmes, nous utilisons d'abord pour les instances représentables graphiquement, i.e. les instances bi-objectives, une analyse visuelle des surfaces de compromis générées par les différents algorithmes. Ces graphiques peuvent aussi nous aider à comprendre les valeurs de la métrique C qui est une métrique relative, qui compare deux algorithmes.

7.4.5.4 Comparaison des différentes variantes de m-ACO

Analyse de la métrique C

Dans cette section nous comparons les cinq variantes de m-ACO suivant la métrique C. Les tableaux 7.1 et 7.2 représentent les résultats de cette métrique sur respectivement les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs. Chaque ligne de ces tableaux affiche successivement les noms des deux variantes comparées, et les valeurs minimales, moyennes et maximales de la métrique C parmi dix exécutions des deux variantes.
En comparant les différentes variantes sur les instances avec 250 objets (tableau 7.1), nous constatons que NV-m-ACO(1,m) trouve les meilleurs résultats. En effet, les valeurs de la métrique C obtenues par NV-m-ACO(1,m) sont toujours supérieures à celles trouvées par les autres variantes. Plus particulièrement, certaines solutions retournées par les variantes m-ACO1(m,m+1) et m-ACO2(m,m+1) sont dominées par NV-m-ACO(1,m), alors que pour la plupart des cas aucune solution de NV-m-ACO(1,m) n'est dominée par d'autres de ces variantes, i.e. les cas où les valeurs de la métrique C sont égales à 0.
La différence des résultats n'est pas aussi importante pour les deux autres variantes

m-ACO3(1,1) et m-ACO4(1,m), puisque ces variantes trouvent des solutions qui dominent d'autres solutions de NV-m-ACO(1,m). Cependant, les valeurs de la métrique C trouvées par NV-m-ACO(1,m) sont supérieures ou égales à celles trouvées par ces deux variantes.
Pour les instances avec 500 objets (tableau 7.2), nous remarquons que les résultats de NV-m-ACO(1,m) sont encore plus élevés que les autres variantes. En effet, cette variante trouve des valeurs de la métrique C supérieures à celles trouvées pour les instances avec 250 objets.
En comparant les autres variantes avec NV-m-ACO(1,m), nous trouvons qu'elles trouvent pour la plupart des cas des valeurs de la métrique C égales ou proches de 0. Par contre,NV-m-ACO(1,m) trouvent des valeurs supérieures de C, égales parfois même à 1 pour l'instance la plus difficile avec 4 objectifs. Pour ces cas, toutes les solutions de NV-m-ACO(1,m) dominent toutes les solutions retournées par la variante comparée. Il est à noter que pour ces instances aussi les deux variantes m-ACO4(1,m) et surtout m-ACO3(1,1) trouvent des résultats assez bons.

Nous concluons cette section par noter que ces résultats de la métrique C confirment ce que nous avons constaté en analysant visuellement les surfaces de compromis retournées par les différentes variantes pour les instances bi-objectives. De plus, les résultats de NV-m-ACO(1,m) s'améliorent encore plus, pour la plupart des cas, pour les instances les plus difficiles avec 3 et 4 objectifs.

Variantes comparées	250.2			250.3			250.4		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
C(1,2)	0	0,03125	0,25	0	0,0153	0,0909	0	0,0071	0,0714
C(2,1)	0	0	0	0	0,0177	0,0725	0	0,0004	0,0020
C(1,3)	0,1341	0,1951	0,2683	0	0,0003	0,0020	0,0009	0,001	0,0011
C(3,1)	0	0,0069	0,0556	0	0	0	0	0,0002	0,002
C(1,4)	0	0,0052	0,0278	0	0,0001	0,0013	0	0,0006	0,0012
C(4,1)	0	0,1071	0,2143	0	0,0089	0,0250	0	0,0059	0,0163
C(1,NV)	0	0,0019	0,0097	0	0,0001	0,0009	0	0,0001	0,0006
C(NV,1)	0	0,0009	0,0085	0	**0,0416**	**0,1277**	0	**0,0214**	**0,0387**
C(2,3)	0	0	0	0	0	0	0	0	0
C(3,2)	0	0	0	0	0,1647	0,4118	0,0667	0,1303	0,3333
C(2,4)	0	0	0	0	0	0	0	0	0
C(4,2)	0	0	0	0	0,0882	0,3529	0	0,02667	0,2

C(2, NV)	0	0	0	0	0	0	0	0	0
C(NV,2)	0	0	0	0	**0,0235**	**0,1176**	0	**0,0626**	**0,2**
C(3,4)	0	0,0119	0,0278	0,0013	0,0055	0,0108	0	0,0009	0,0036
C(4,3)	0	0,0109	0,0482	0	0	0	0	0	0
C(3, NV)	0	0,001	0,0103	0	0,0004	0,0019	0	0,0002	0,0011
C(NV,3)	0	**0,0072**	**0,0388**	0	**0, 0010**	**0,0057**	0	**0,0003**	**0,003**
C(4, NV)	0	0,0019	0,0105	0	0,0002	0,0009	0	0	0,0003
C(NV,4)	0	**0,0264**	**0,0833**	**0,0012**	**0,0045**	**0,0105**	0	**0,002**	**0,0104**

Tableau. 7.1 - Comparaison des variantes de m-ACO suivant la métrique C sur les instances avec 500 objets et 2, 3 et 4 objectifs

Variantes comparées	500.2			500.3			500.4		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
C(1,2)	0	0,1	1	0	0,0158	0,0909	0,4167	0,6276	0,7778
C(2,1)	0	0,0001	0,0006	0	0,0003	0,002	0	0	0
C(1,3)	0	0,0032	0,0213	0	0	0	0	0	0
C(3,1)	0	0,0168	0,0909	0	0,0535	0,1	0,0046	0,0392	0,0951
C(1,4)	0	0,0013	0,0132	0	0,0004	0,0015	0	0	0,0004
C(4,1)	0	0,1495	0,8889	0	0,0117	0,0370	0	0,0081	0,0186
C(1, NV)	0	0,0025	0,025	0	0,0002	0,002	0	0	0
C(NV,1)	0	0,0444	0,4444	0	0,0618	0,2	0,0046	0,034	0,1021
C(2,3)	0	0	0	0	0	0	0	0	0
C(3,2)	0	0	0	0	0,0548	0,2941	0,8462	0,9231	1
C(2,4)	0	0	0	0	0	0	0	0	0
C(4,2)	0	0,1	1	0	0,0248	0,1818	0,7692	0,9297	1
C(2, NV)	0	0	0	0	0	0	0	0	0
C(NV,2)	0	0	0	0	**0,1087**	**0,3125**	1	**1**	**1**
C(3,4)	0	0,0331	0,1053	0	0,0136	0,0392	0,0004	0,0038	0,0095
C(4,3)	0	0,0537	0,3553	0	0	0	0	0	0,0002
C(3, NV)	0	0,0023	0,0227	0	0,0021	0,0137	0	0,0001	0,0012
C(NV,3)	0	0,0064	0,0213	0	0,0022	0,0111	0	0,0003	0,0023
C(4, NV)	0	0,0099	0,0581	0	0	0	0	0	0
C(NV,4)	0	0,0189	0,0714	0	0,0125	0,036	0	0,0029	0,0073

Tableau. 7.2 - Comparaison des variantes de m-ACO suivant la métrique C sur les instances avec 500 objets et 2, 3 et 4 objectifs

7.4.5.5 Comparaison de NV-m-ACO(1,m) avec les algorithmes évolutionnaires

Nous comparons dans cette section la nouvelle variante NV-m-ACO(1,m), qui trouve les meilleurs résultats comparée aux autre approches, avec des algorithmes évolutionnaires de l'état de l'art.

Analyse de la métrique C

Dans cette section nous comparons la nouvelle variante NV-m-ACO avec les algorithmes évolutionnaires suivant la métrique C. Les tableaux 7.4 et 7.5 comparent NV-m-ACO, sur respectivement les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs, avec les résultats des algorithmes VEGA, SPEA, NSGA, HLGA et NPGA. Le tableau 5.6 présente les résultats de NV-m-ACO et les algorithmes VEGA, SPEA, SPEA2, NSGA, NSGAII, HLGA et NPGA sur les instances avec 750 objets avec 2, 3 et 4 objectifs. Chaque ligne de ces tableaux affiche successivement les noms des deux algorithmes comparés, et les valeurs minimales, moyennes et maximales de la métrique C parmi dix exécutions des deux algorithmes.

En comparant NV-m-ACO avec tous les algorithmes évolutionnaires, nous constatons qu'elle est nettement meilleure sur toutes les instances testées. En effet, les solutions de NV-m-ACO dominent largement les solutions retournées par les autres algorithmes. De plus, aucune solution de tous les AE comparés ne domine une autre de NV-m-ACO puisque les valeurs de la métrique C pour tous ces algorithmes et pour toutes les instances sont égales à 0.

Nous constatons aussi que plus sont difficiles et grandes les instances, mieux sont les résultats de NV-m-ACO. En effet, pour les instances avec 500 objets, les valeurs de la métrique C de NV-m-ACO avec tous les AE sont égales ou proches de 1 sauf pour SPEA qui est l'algorithme le plus performant parmi la liste des AE comparés pour ces instances. Pour les instances avec 750 objets, les valeurs C de NV-m-ACO avec les AE sont égales à 1 sauf pour SPEA, SPEA2 et NSGAII. En effet, ces deux derniers algorithmes sont parmi les algorithmes évolutionnaires les plus performants et les plus connus dans la littérature. Les résultats de ces algorithmes sont disponibles que pour les instances avec 750 objets.

Algorithmes comparés	**250.2**			**250.3**			**250.4**		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(NV, VEGA)	0,7895	0,9105	0,9474	0,4810	0,8539	0,9806	0,3004	0,6617	0,9009
C(VEGA, NV)	0	0	0	0	0	0	0	0	0
C(NV , SPEA)	0,1333	0,17	0,2	0,0485	0,1676	0,2283	0,1782	0,2571	0,3228

C(SPEA, NV)	0	0	0	0	0	0	0	0	0
C(NV , NSGA)	0,6154	0,6731	0,7308	0,1743	0,6321	0,8129	0,3374	0,5574	0,7756
C(NSGA, NV)	0	0	0	0	0	0	0	0	0
C(NV , HLGA)	0,6923	0,7308	0,8462	0,8793	0,9841	1	0,8947	0,9787	1
C(HLGA, NV)	0	0	0	0	0	0	0	0	0
C(NV, NPGA)	0,9444	0,9833	1	0,4510	0,7632	0,9639	0,2780	0,6259	0,8338
C(NPGA, NV)	0	0	0	0	0	0	0	0	0

Tableau. 7.4 - Comparaison de NV-m-ACO(1,m) avec les algorithmes évolutionnaires suivant la métrique C sur les instances avec 250 objets et 2, 3 et 4 objectifs.

Algorithmes comparés	**500.2**			**500.3**			**500.4**		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(NV, VEGA)	1	1	1	0,9855	0,9986	1	1	1	1
C(VEGA, NV)	0	0	0	0	0	0	0	0	0
C(NV, SPEA)	0,3514	0,5795	0,7188	0,3707	0,5321	0,6946	0,6688	0,7598	0,8620
C(SPEA, NV)	0	0	0	0	0	0	0	0	0
C(NV, NSGA)	1	1	1	0,9489	0,9790	1	0,9975	0,999	1
C(NSGA, NV)	0	0	0	0	0	0	0	0	0
C(NV , HLGA)	1	1	1	1	1	1	1	1	1
C(HLGA, NV)	0	0	0	0	0	0	0	0	0
C(NV , NPGA)	1	1	1	1	1	1	1	1	1
C(NPGA, NV)	0	0	0	0	0	0	0	0	0

Tableau. 7.5 - Comparaison de NV-m-ACO(1,m) avec les algorithmes évolutionnaires suivant la métrique C sur les instances avec 500 objets et 2, 3 et 4 objectifs.

En comparant la nouvelle variante NV-m-ACO(1,m) avec NSGAII et SPEA2 sur les instances avec 750 objets, nous constatons qu'ils sont très proches pour les trois instances testées avec un très léger avantage pour SPEA2.

7.4.5.6 Analyse globale

Dans cette section nous essayons de tirer quelques conclusions générales résumant toutes les analyses réalisées dans cette étude expérimentale.
Dans la première partie de cette étude, nous avons comparé les variantes proposées de m-ACO (4 variantes de l'état de l'art et une nouvelle variante). Nous rappelons que les deux variantes m-ACO2(m,m+1), m-ACO3(1,1) reprennent respectivement les schémas généraux des algorithmes MOACO de la littérature CompetAnts [113] et MACS [112]. Les deux autres variantes m-ACO1(m,m+1), m-ACO4(1,m) représentent des approches relativement nouvelles, et NV-m-ACO(1,m) représente la nouvelle variante que nous avons proposée.
En comparant ces différentes variantes suivant la métrique C, nous avons trouvé que la nouvelle variante NV-m-ACO(1,m) trouve les meilleurs résultats.
Suivant la métrique C, le front de NV-m-ACO(1,m) domine les fronts de toutes les autres variantes. En effet, les valeurs de la métrique C de NV-m-ACO(1,m) sont toujours supérieures aux autres variantes. De plus, les variantes m-ACO3(1,1) et m-ACO4(1,m) trouvent des valeurs de C assez bonnes.
En comparant les deux variantes m-ACO1(m,m+1) et m-ACO2(m,m+1), qui trouvent globalement les résultats les moins bons, nous remarquons que m-ACO1(m,m+1) est meilleure que m-ACO2(m,m+1) surtout pour les instances les plus difficiles. Nous rappelons que la seule différence entre ces deux variantes est dans la colonie m+1 qui utilise, dans la première variante, la trace de phéromone correspondant à l'objectif choisi aléatoirement à chaque étape de construction, alors que pour la deuxième, elle utilise une agrégation de toutes les traces de phéromone. Ainsi, il semble que l'utilisation d'une agrégation des traces de phéromone n'est pas un bon choix, pour cet algorithme et pour le MOKP.
La variante m-ACO3(1,1), qui reprend l'idée de base de l'algorithme MACS [112], trouve des résultats qui sont assez bons. D'ailleurs, dans [119] les auteurs, qui ont comparé plusieurs algorithmes MOACO sur le problème du voyageur de commerce bi-objectifs, ont trouvé que cet algorithme trouve parmi les meilleurs résultats. Cette variante utilise les solutions du front Pareto pour la mise à jour de phéromone.

Les deux variantes m-ACO4(1,m) et NV-m-ACO(1,m) utilisent une idée relativement nouvelle. En effet, elles utilisent une seule colonie de fourmis et plusieurs structures de phéromone. De plus, les fourmis considèrent à chaque étape de construction une structure de phéromone choisie aléatoirement.
NV-m-ACO(1,m) n'utilise de plus, par rapport à m-ACO4(1,m), que la mise à jour de phéromone sur le front Pareto. Ainsi, il est clair l'apport de cette mise à jour sur la qualité des

solutions générées. Ceci peut expliquer aussi la qualité des solutions assez bonne de m-ACO3(1,1) et aussi de MACS sur le problème du voyageur de commerce [119].
Enfin, nous pouvons conclure pour cette étude que la variante NV-m-ACO(1,m) trouve les meilleurs résultats non seulement grâce à la nouvelle idée qu'elle utilise mais aussi grâce à la mise à jour de phéromone sur le front Pareto. En effet, NV-m-ACO(1,m) et m-ACO3(1,1) appartiennent à la classe des approches Pareto . Ceci peut rejoindre les études qui ont été réalisées sur les algorithmes évolutionnaires dans la littérature qui classent les approches Pareto parmi les meilleures approches dans l'optimisation multi-objectif.
Dans la deuxième partie de cette étude, nous avons comparé NV-m-ACO(1,m) avec plusieurs algorithmes évolutionnaires de l'état de l'état de l'art. Nous avons trouvé que NV-m-ACO(1,m) obtient des fronts Pareto qui dominent toujours et largement tous les fronts des AE sur toutes les instances testées. De plus, les fronts Pareto générés par ces AE ne dominent jamais ceux de NV-m-ACO(1,m). Il est à noter que tous les AE comparés sont des approches Pareto, sauf VEGA et HLGA, qui trouvent d'ailleurs les résultats les moins bons.

Algorithmes comparés	**750.2**			**750.3**			**750.4**		
	Min	**Moy**	**Max**	**Min**	**Moy**	**Max**	**Min**	**Moy**	**Max**
C(NV, VEGA)	1	1	1	1	1	1	-	-	-
C(VEGA, NV)	0	0	0	0	0	0	-	-	-
C(NV , SPEA)	0,2143	0,6641	0,9545	0,17	0,245	0,2867	0,7629	0,826	0,8857
C(SPEA, NV)	0	0	0	0	0	0	0	0	0
C(NV , SPEA2)	0,108	0,164	0,196	0,08	0,1277	0,1567	0,5029	0,56	0,6657
C(SPEA2, NV)	0	0	0	0	0	0	0	0	0
C(NV , NSGA)	1	1	1	1	1	1	-	-	-
C(NSGA, NV)	0	0	0	0	0	0	-	-	-
C(NV , NSGAII)	0,092	0,2192	0,268	0,1467	0,1957	0,2367	0,4886	0,5769	0,6371
C(NSGAII, NV)	0	0	0	0	0	0	0	0	0
C(NV , HLGA)	1	1	1	1	1	1	1	1	1
C(HLGA, NV)	0	0	0	0	0	0	0	0	0
C(NV , NPGA)	1	1	1	1	1	1	1	1	1
C(NPGA, NV)	0	0	0	0	0	0	0	0	0

Tableau. 7.6 - Comparaison de NV-m-ACO(1,m) avec les algorithmes évolutionnaires suivant la métrique C sur les instances avec 750 objets et 2, 3 et 4 objectifs

7.4.6 Conclusion

Nous avons appliqué, dans ce chapitre, les variantes de l'algorithme générique m-ACO sur le problème du sac à dos multi-objectif. Nous avons testé et comparé les différentes variantes sur plusieurs instances benchmarks du MOKP. Pour comparer ces différentes variantes, nous avons utilisé la métrique de performance appelée la métrique C. Les résultats trouvés par cette métriques étaient en faveur de la nouvelle variante proposée NV-m-ACO(1,m). NV-m-ACO(1,m) utilise une idée relativement nouvelle puisqu'elle utilise une seule colonie de fourmis et m structures de phéromone, i.e. une structure pour chaque objectif. De plus, les fourmis considèrent à chaque étape de construction une structure de phéromone choisie aléatoirement. Cependant, nous avons constaté que cette nouvelle variante trouve les meilleurs résultats grâce à cette nouvelle idée et aussi grâce à la mise à jour de phéromone sur le front Pareto.

En effet, nous avons pu conclure que pour les algorithmes MOACO, les approches Pareto trouvent généralement les meilleurs résultats.

En comparant NV-m-ACO(1,m) avec plusieurs algorithmes évolutionnaires de l'état de l'art, nous avons trouvé que NV-m-ACO(1,m) est nettement meilleur sur toutes les instances testées.

Conclusion et Perspectives

Dans ce mémoire, nous avons investigué les capacités de l'optimisation par colonies de fourmis. Nous avons étudié les problèmes d'optimisation multi-objectif et nous avons proposé une taxonomie des algorithmes fourmis présentés dans la littérature pour résoudre ce type de problèmes. Cette taxonomie classe les différents algorithmes suivant quatre points : la définition des structures phéromone, du facteur phéromone, du facteur heuristique et des solutions à récompenser.

Nous avons présenté, par la suite, une étude de différentes stratégies phéromonales que sont appliquées sur le problème du sac à dos multidimensionnel (MKP) pour le cas uni-objectif. Nous avons présenté trois versions différentes d'un algorithme générique. Les trois versions diffèrent suivant les composants sur lesquels les traces de phéromone sont déposées : Vertex-AK où la phéromone est déposée sur les objets, Path-AK où la phéromone est déposée sur les couples d'objets sélectionnés consécutivement, et Edge-AK où la phéromone est déposée sur les paires d'objets sélectionnés dans une même solution.

Les expérimentations ont montré que Path-AK obtient de bien moins bons résultats que Vertex-AK et Edge-AK et que Edge-AK obtient de résultats légèrement meilleurs que Vertex-AK. Cependant, Edge-AK prend plus de temps CPU que Vertex-AK lors de l'exécution, puisque la phase de mise à jour de phéromone est de complexité quadratique pour Edge-AK alors qu'elle est de complexité linéaire pour Vertex-AK. Nous avons étudié aussi l'influence des paramètres de l'algorithme fourmi sur la qualité des solutions construites ainsi que sur la similarité de ces solutions.

Nous nous sommes basés sur cette étude, pour le cas multi-objectif, pour choisir la stratégie phéromonale appropriée, et aussi pour choisir les valeurs des paramètres de l'algorithme ACO.

Nous avons présenté par la suite un algorithme générique, appelé m-ACO, basé sur l'optimisation par colonies de fourmis pour résoudre des problèmes d'optimisation multi-objectif.

Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de structures de phéromone. En effet, nous considérons que le nombre de colonies de fourmis peut être différent du nombre de structures de phéromone. Cet algorithme permet de tester et de comparer, dans un même cadre, différentes approches existantes ainsi que de nouvelles

approches sans se soucier du schéma ACO considéré ni des détails d'implémentation. Nous avons présenté quatre variantes de cet algorithme et nous avons proposé une nouvelle variante de cet algorithme utilisant une approche relativement nouvelle.
Nous avons appliqué et testé les différentes variantes, par la suite, sur le problème du sac à dos multidimensionnel multi-objectif. Après une étude comparative des différentes variantes suivant différentes mesures de performance, nous avons trouvé que la nouvelle variante NV-m-ACO(1,m) trouve globalement les meilleurs résultats. Cette variante utilise une idée relativement nouvelle puisqu'elle utilise une seule colonie de fourmis et une structure de phéromone pour chaque objectif, et les fourmis considèrent aléatoirement à chaque étape de construction un objectif à optimiser. *De plus, la mise à jour de phéromone se fait non seulement sur les meilleures solutions trouvées pour chaque objectif, comme dans la variante m-ACO4(1,m), mais aussi sur le front Pareto.*
Nous avons pu constater l'apport de la mise à jour de phéromone sur le front Pareto en comparant les deux variantes m-ACO4(1,m) et NV-m-ACO(1,m). Nous avons remarqué aussi que la variante m-ACO3(1,1), qui utilise une approche Pareto, trouve des résultats assez bons et proches pour plusieurs instances de NV-m-ACO(1,m). En effet, l'algorithme MACS, que m-ACO3(1,1) reprend l'idée de base, trouve aussi parmi les meilleurs résultats comparé avec plusieurs algorithmes MOACO de l'état de l'art dans [151] pour le problème du voyageur de commerce. Ainsi, nous avons pu conclure que pour les algorithmes MOACO, tout comme les algorithmes évolutionnaires, les approches Pareto semblent trouver les meilleurs résultats.
Nous avons comparé par la suite la variante NV-m-ACO(1,m) avec plusieurs algorithmes évolutionnaires, qui utilisent pour la plupart une approche Pareto. Nous avons trouvé que NV-m-ACO(1,m) est nettement meilleure sur toutes les instances testées.

Nos futurs travaux de recherche essayeront d'appliquer la nouvelle variante de l'algorithme m-ACO à d'autres problèmes d'optimisation mutli-objectif. Nous nous sommes focalisés durant ce mémoire sur des problèmes de type sac à dos. Nous envisageons de traiter d'autres problèmes comme le problème du voyageur de commerce multi-objectif par exemple.
Nous avons proposé une nouvelle variante de l'algorithme générique m-ACO. D'autres variantes peuvent être proposées pour cet algorithme qui, soit proposent de nouvelles idées, soit reprennent d'autres algorithmes MOACO de l'état de l'art.
Nous avons comparé nos approches avec les algorithmes évolutionnaires les plus connus et les plus performants de l'état de l'art. Une autre perspective serait de nous comparer avec

d'autres algorithmes basés sur d'autres méta-heuristiques de l'état de l'art comme le recuit simulé ou la recherche tabou.
Enfin, il serait intéressant aussi d'appliquer notre approche sur un problème réel qui pourrait tester l'efficacité de cette approche quand elle est employée réellement.

Contributions Scientifiques :

Les travaux réalisés depuis le début de ce mémoire font l'objet des contributions suivantes :

- Saha, A. , Maamri R. A New Pheromonal Strategy in Ant Algorithm for Multi-Objective Knapsack Problems. MASAUM Journal of Basic and Applied Sciences (MJBAS) Vol.1 No.1 (August2009).
  http://www.masaumnet.com/archives/mjbas/volume1/issue1/mjbas010122.pdf
- Saha, A. , Maamri R. Ant Algorithm for Multi-Objective Knapsack Problem. International Conference On Applied Informatics (ICAI09) November 15-17 , 2009. Proceeding Volume 1 Computer Business Intelligence. Page 416-422. Depot Legal: 4693-2009 ISBN 978-9947-0-2763-9.
- Saha, A. , Maamri R. Résolution des Problèmes Multi Objectifs à Base de Colonies de Fourmi. Doctoriales STIC09**.** Université Mohamed Boudiaf de M'sila, 7 et 8 décembre 2009.

Bibliographie

[1] Alain Berro ; Optimisation Multiobjectif et Stratégie d'évolution en environnement Dynamique (thèse de Doctorat) ; 18 Décembre 2001 ; Université des sciences sociales ToulouseI ; page 14, 27, 29.

[2] Yann Collette, Patrick Siarry ; Optimisation Multiobjectif ; Septembre 2002 ; Edition Eyrolles, 75240 Paris Cedex 05, France ; Page 1, 1, 2, 17, 18, 19, 21, 22, 23, 41, 42, 43, 44, 46.

[3] M. Caldora Costa ; Optimisation de Diapositifs Electromagnétiques dans un Contexte d'analyse par la Méthode des Eléments Finis ; 17 Juillet 2001 ; L.E.G- Laboratoire d'électrotechnique de Grenoble ; Page 10.

[4] Johan Andersson ; Multiobjective optimization in Engineering Design, Applications to Fluid Power System (thèse de Doctorat); 2001; Université de Linköping, Sweden; Page 29.

[5] Mostapha Benhenda ; http://www.eleves.ens.fr/home/benhenda.html

[6] D. Gaertner ; Natural Algorithms for Optimisation Problems, Outsourcing Report; 16 Janvier 2004.

[7] Vincent Barichard ; Approches Hybrides pour les Problèmes Multiobjectifs (thèse de Doctorat) ; 24 Novembre 2003 ; Université d'Angers ; Page 9, 11,37, 40.

[8] Tollari Sabrina ; 2003 ; http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node4.html

[9] Vincent Barichard & Jin-Kao Hoa ; Un Algorithme Hybride pour le Problème de Sac à Dos Multi-objectifs ; 2002 ; Article d'Actes JNPC'02 ; Page 1.

[10] S.-Y. Shin, B.-T. Zhang et S.-S. Jun; Solving Traveling Salesman Problems Using Molecular Programming; 1999.

[11] Vincent Barichard & Jin-Kao Hoa ; Un Algorithme Hybride pour le Problème de Sac à Dos Multi-objectifs ; 2002 ; Article d'Actes JNPC'02 ; Page 1.

[12] Chu, P.C., & Beasley, J.E. 1998. A genetic algorithm for the multidimentional knapsack problem. Journal of heuristics, 4, 63-86.

[13] M. Yokoo, T. Suyama, et H. Sawada ; Solving satisfiability problems using field programmable gate arrays: first results; In 2nd International Conference on Principles and Practice of Constraint Programming (CP'96); page 497–509, Springer; 1996.

[14] T. Suyama, M. Yokoo, H. Sawada, et A. Nagoya ; Solving satisfiability problems using reconfigurable hardware; IEEE Transactions on VLSI Systems, 9(1):109–116; 2001.

[15] P. Zhong, P. Ashar, S. Malik, et M. Martonosi ; Using reconfigurable computing techniques to accelerate problems in the CAD domain: a case study with Boolean satisfiability; In 35th Design Automation Conference (DAC), page 194–199; 1998.

[16] M. Davis et H. Putnam ; A computing procedure for quantification theory; Journal of the ACM, (7):201–215; 1960.

[17] P. Zhong, M. Martonosi, P. Ashar, et S. Malik ; Using configurable computing to accelerate Boolean satisfiability; IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 18(6):861–868; 1999.

[18] M. Platzner et G. D. Micheli ; Acceleration of satisfiability algorithms by reconfigurable hardware; In 8th International Workshop on Field-Programmable Logic and Applications (FPL'98), page 69–78; Springer; 1998.

[19] O. Mencer, M. Platzner, M. Morf, et M. J. Flynn; Object-oriented domain specific compilers for programming FPGAs; IEEE Transactions on VLSI Systems, 9(1):205–210, 2001.

[20] M. Abramovici et J. T. De Sousa ; A SAT solver using reconfigurable ardware and virtual logic; Journal of Automated Reasoning, 24(1–2):5–36; 2000.

[21] M. Abramovici et D. Saab ; Satisfiability on reconfigurable hardware; In 7th International Workshop on Field-programmable Logic and Applications (FPL'97), page 448–456; Springer; 1997.

[22] C. Blessl et M. Platzner ; Instance-Specific Accelerators for Minimum Covering ; In the Journal of Supercomputing , 26, 109-129; 2003; Page 110, 111, 112.

[23] S. Prestwich et C. Quirke ; Boolean and Pseudo-Boolean Models for Scheduling; Page 1.

[24] C. M. Li; Integrating Equivalency Reasoning into Davis-Putnam Procedure; Seventeenth National Conference on Artificial Intelligence; Austin, Texas, USA; 2000; page 291–296.

[25] P. Baumgartner et F. Massacci; The Taming of the (X)OR; First International Conference on Computational Logic, Stream on Automated Deduction: Putting Theory into Practice, Lecture Notes in Artificial Intelligence vol. 1861; Springer-Verlag; 2000; page 508–522.

[26] J. Whittemore, J. Kim et K. Sakallah; SATIRE: A New Incremental Satisfiability Engine; Thirty-Eighth Design Automation Conference; 2001; page 542–545.

[27] M. R. Dransfield, V. W. Marek et M. Truszczynski. Satisfiability and van der Waerden Numbers; Sixth International Conference on Theory and Applications of Satisfiability Testing; Portofino, Italy; 2003; page 325-336.

[28] H. E. Dixon, M. L. Ginsberg et A. J. Parkes; Likely Near-term Advances in SAT Solvers; Workshop on Microprocessor Test and Verification; Austin, Texas, USA; 2002.

[29] A. Ramani, F. A. Aloul, I. L. Markov et K. A. Sakallah ; Breaking Instance-Independent Symmetries in Exact Graph Coloring; 2004; Page 2.

[30] G. J. Chaitin et. Al; Register allocation via coloring; in Computer Languages; 6:47-57; 1981.

[31] Artificial Intelligence: Introduction to local search; Page 9.

[32] Rémy-Robert, Alexandre Joseph ; Systèmes Interactifs d'Aide à l'Élaboration de Plannings de Travail de Personnel Contraintes, Aide à la Décision, Représentation Combinatoire des Préférences, Équité et Résolution par Décomposition Arborescente et par Consistance ; 07 Novembre 2003 ; Université Joseph Fourier-Grenoble1, Science et Géographie ; Page 28.

[33] J. Andersson ; A Survey for Multiobjective Optimization in engineering Design; Rapport technique, LiTH-IKP-R-1097; 2000.

[34] I. Othmani. Optimisation multicritère : Fondements et Concepts. PhD thesis, Université de Grenoble, 1998.

[35] M. Ehrgott. Multicriteria optimization. In Lecture Notes in Economics and Mathematical Systems, volume 491. Springer, 2000.

[36] Y. Collette et P. Siarry. Optimisation multiobjectif. Eyrolles, 2002.

[37] E.L. Ulungu and J. Teghem. Multi-objective combinatorial optimization: a survey. Journal of Multi-Criteria Decision Analysis, 3:83{104, 1994.

[38] K. Miettinen. Nonlinear multiobjective optimization. Kluwer Academic Publishers,1999.

[39] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum, 22:425-460, 2000.

[40] Deb, 2001 K. Deb. Multi-objective optimization using evolutionary algorithms. John Wiley, 2001.

[41] A. Charnes andW.W. Cooper. Management models and industrial applications of linear programming. Wiley, New York, 1961.

[42] C. Romero. Handbook of critical issues in goal programming. Oxford, UK: Pergamon Press, 1991.

[43] C.A. Coello Coello. An updated survey of G.A. based multiobjective optimization techniques. Technical report, Lania-RD-98-08, Xalapa, Veracruz, Mexico, 1998.

[44] Ph. Vincke. Multicriteria decision aid. J. Wiley, New York, 1992.

[45] D. Bouyssou, E. Jacquet-Lagr7ze, P. Perny, R. Slowinski, D. Vanderpooten, and P. Vincke. Aiding decisions with multiple criteria. Kluwer, 2001.

[46] E.L. Ulungu. Optimisation combinatoire multicritère : détermination de l'ensemble des solutions efficaces et méthodes interactives. PhD thesis, Faculté des Sciences, Université de Mons-Hainault, 1993.

[47] E.L. Ulungu and J. Teghem. The two-phases method: an efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of Computing and Decision Sciences, 20(2):149{165, 1994.

[48] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. Journal of Global Optimization, 12:139-155, 1998.

[49] C.R. Reeves. Modern heuristic techniques for combinatorial problems. McGraw Hill, 1995.

[50] J.K. Hao, P. Galinier, et M. Habib. M□ etaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence Artificielle, 13(2):283-324, 1999.

[51] D. Corne, M. Dorigo, and F. Glover. New ideas in optimization. McGraw Hill, 1999.

[52] M. Pirlot and J. Teghem. Optimisation approchée en recherche opérationnelle (Traité IC2, Série Informatique et systèmes d'information). Hermès Sciences, 2002.

[53] M. Pirlot and J. Teghem. Résolution de problèmes de RO par les métaheuristiques (Traité IC2, série Informatique et systèmes d'information). Hermès Sciences, 2003.

[54] E. Bonomi et J.-L. Lutton ; le recuit simulé pour la science ; numéro 129, pages **68-77 ; Juillet 1988.**

[55] M. O'Keeffe et M. Ô Cinnéide ; A Stochastic Approach to automated Design Improvement; 2003.

[56] Le Recuit Simulé ; 17 Novembre 2004.

[57] F. Glover; Artificial Intelligence, heuristic frameworks and tabu search; Managerial and decision economics; volume 11, pages 365-375; 1990.

[58] D. De Werra; heuristics for graph coloring; computing Suppl; volume 7, pages 191-208; 1990.
[59] S. Kemmoé , L. Deroussi, M. Gourgand et A. Quilliot ; Des Essaims Particulaires Efficaces Pour l'optimisation Combinatoire ; Congrès du GDR MACS, Pôle STP ; Octobre 2004.
[60] T. Back, D.B. Fogel, Z. Michalewicz, and T. Baeck. Handbook of Evolutionary Computation. Institute of Physics Publishing and Oxford University Press, 1997.
[61] J.H. Holland. Adaptation in natural and artificial systems. PhD thesis, University of Michigan Press, 1975.
[62] D.E. Goldberg. Genetic algorithms for search, optimization, and machine learning.Reading, MA: Addison-Wesley, 1989.
[63] H-P. Schwefel. Numerical optimization of computer models. Wiley, Chichester, 1981.
[64] D. Fogel. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence (second edition). IEEE Press, 2000.
[65] I. Charon, A. Germa, O. Hudry, Méthodes d'Optimisation Combinatoire ; 1996.
[66] N. Srinivas and K. Deb. Multiobjective optimization using non dominated sorting in genetic algorithms. Evolutionary Computation, 2(3):221{248, 1994.
[67] K.A. De Jong. An analysis of the behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan, 1975.
[68] H. Tamaki, M. Mori, M. Araki, Y. Mishima, and H. Ogai. Multicriteria optimization by genetic algorithms: A case of scheduling in hot rolling process. In Proceedings of APORS'94, pages 374{381. World Scientific Publishing, 1994.
[69] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm for multi-objective optimization: NSGA-II. In Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI), pages 849-858, 2000.
[70] H. Ishibuchi and T. Murata. Multi-objective genetic local search algorithm. In Proceedings of IEEE International Conference on Evolutionary Computation (ICEC'96), pages 119-124. IEEE, 1996.
[71] G.T. Parks and I. Miller. Selective breeding in a multiobjective genetic algorithm. In Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V), pages 250{259. Springer, 1998.
[72] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation, 3:257-271, 1999.
[73] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, pages 41{49. Lawrence Erlbaum, 1987.
[74] T. Blickle. Theory of evolutionary algorithms and application to system-synthesis. PhD thesis, Swiss Federal Institute of Technology (Zurich), 1996.
[75] E-G. Talbi. Une taxinomie des m2taheuristiques hybrides. Dans ROADEF'2000, 2000.
[76] M.P. Hansen. Tabu search for multiobjective optimization: Mots. In Proceedings of 13th International Conference on MCDM, pages, 1997.

[77] X. Gandibleux, N. Mezdaoui, and A. Freville. A multiobjective tabu search procedure to solve combinatorial optimization problems. In Lecture Notes in Economics and Mathematical Systems, volume 455, pages 291{300. Springer, 1997.

[78] P. Czyzak and A. Jaszkiewicz. A Pareto simulated annealing - a metaheuristic for multiple-objective combinatorial optimization. Journal of Multi-Criteria Decision Analysis, 7(1):34-47, 1998.

[79] D.W. Corne and J.D. Knowles. M-paes: a memetic algorithm for multiobjective optimization. In Proceedings of the 2000 Congres on Evolutionary Computation, pages 325-332, 2000.

[80] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: empirical results. Evolutionary Computation Journal, 8(2):125-148, 2000.

[81] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms: a comparative case study. In Lecture Notes in Computer Science, pages 292-301. Springer, 1998.

[82] J.N. Morse. Reducing the size of the nondominated set: pruning by clustering. Computers and Operations Research, 7:55-66, 1980.

[83] Guntsch, M., & Middendorf, M. A population based approach for ACO. Page 7281 of : In S. Cagnoni, J. Gottlieb, E. Hart M. Middendorf, & G. R. Raidl, editors (eds), EvoWorkshops.

[84] Jong, K.A. De. 1975. An analysis of the behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan.

[85] Garcia-Martinez, C., Cordon, O., & Herrera, F. 2007. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for bi-criteria tsp. Pages 116-148 of : European Journal of Operational Research.

[86] Angus, D. 2007. Crowding population-based ant colony optimisation for the multi-objective travelling salesman problem. Pages 333-340 of : In 2007 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM 2007). IEEE.

[87] V. Maniezzo, M. Dorigo and A. Colorni, The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1) :29-41, 1996.

[88] L. Bianchi, L.M. Gambardella, M.Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In Proceedings of PPSN-VII, Seventh Inter17 national Conference on Parallel Problem Solving from Nature Science. Springer Verlag, Berlin, Germany, 2002

[89] J.L. Bentley, Fast algorithms for geometric traveling salesman problem, ORSA Journal on Computing, vol. 4, pp. 387-411, 1992.

[90] B. Bullnheimer, R.F. Hartl, and C. Strauss, Applying the ant system to the vehicle routing problem, In: Voss

[91] C. Blum, M. Sampels, Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations, in Proceedings of the 2002 congress on Evolutionary Computation, Honolulu, USA

[92] M. den Besten, T. Stützle, M. Dorigo, Ant colony optimization for the total weighted tardiness problem, Parallel Problem Solving from Nature: 6th international conference, September 2000. Springer Verlag.

[93] A. Colorni, M. Dorigo, and V. Maniezzo, Distributed optimization by ant colonies, Proceedings of ECAL'91, European Conference on Artificial Life, Elsevier Publishing, Amsterdam, 1991.

[94] O. Cordon, I. Fernandez de Viana, F. Herrera, L. Moreno, A new ACO model integrating evolutionary computation concepts: the best-worst ant system, in Proceedings of ANTS2000 -from ant colonies to artificial ants, Universitè Libre de Bruxelles

[95] D. Camara, A.A.F. Loureiro, A GPS/ant-like routing algorithm for ad hoc networks, in 2000 IEEE Wireless Communications and Networking Conference, Chicago, USA.

[96] Holldobler and Wilson, Voyage chez les Fourmis. Seuil, 1996.

[97] O. Roux, La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation et à la programmation automatique, thèse de doctorat de l'Université du Littoral Cote d'Opale, 2001.

[98] N. Monmarché, Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation, thèse de doctorat, l'Université de Tours, le 20 décembre 2000.

[99] R. Vander Meer, M. Breed, K.E., E., and M.L., W., editors (1998). *Pheromone Communication in Social Insects*. Westview Press.

[100] Brossut, R. (1996). *Phéromones, la communication chimique chez les animaux*. CNRS editions, Belin.

[101] G. di Caro and M. Dorigo, AntNet: distributed stigmergetic control for communications network , Journal of Artificial Intelligence Research (JAIR), Vol. 9, Pag. 317- 365, 1998.

[102] G. Dicaro and M. Dorigo, Antnet: distributed stigmergetic control for communications networks, Journal of Artificial Intelligence Research, 9 (1998), 317-365.

[103] G. di Caro and M. Dorigo, Mobile agents for adaptive routing, Proceedings of HICSS-31, 1998.

[104] S.Bouri., A. Zeblah, A. Ghoraf, S. Hadjeri, H. Hamdaoui,Ant Colony Optimization to Shunt Capacitor Allocation in Radial Distribution Systems Acta and Electronica et informatic journal Schekoslovacia N°4,Vo5; 2005

[105] R. Meziane, H. Hamdaoui, M. Rahli, and A. Zeblah Structure Optimization of Electrical Power Network Using Ant Colony Approach. Facta Univ. Ser.: Elec. Energ., vol. 16, No. 2, August 2003, pp. 233-250.

[106] R. Ouiddir, M. Rahli, R. Meziane and A. Zeblah. Ant colony Optimization For New Redesign Problem Of Multi-States Electrical Power Systems. Journal Of Electrical Engineering, Vol 55, N° 1-2, 2004, pp 1-7, ISSN 13-35-36-32 FEISTU

[107] Czyzak, P., & Jaszkiewicz, A. 1998. Pareto simulated annealing a metaheuristic technique for multiple-objective combinatorial optimisation. Journal of Multi-Criteria Decision Analysis, 7, 34-47.

[108] Suppapitnarm, A., & Parks, T. 2001. Simulated annealing : an alternative approach to true multiobjective optimization. In : Genetic and Evolutionary Computation Conference.

[109] Gandibleux, X., Mezdaoui, N., & Fréville, A. A multiob-jective tabu search procedure to solve combinatorial optimization problems.

[110] Abdelaziz, F. Ben, & Krichen, S. 1997. A tabu search heuristic for multiple objective knapsack problems. Ructor Research Report RR 28-97.

[111] Jaszkiewicz, A. 2002a. Genetic local search for multiple objective combinatorial optimization. European Journal of Operational Research, 137(1), 50-71.

[112] Jaszkiewicz, A. 2002b. On the Performance of Multiple-Objective Genetic Local Search on the 0/1 Knapsack Problem-A Comparative Experiment. IEEE Transactions on Evolutionary Computation, 6(4), 402-412.

[113] Barichard, V., & Hao, J.K. 2003. Genetic tabu search for the multi-objective knapsack problem. Tsinghua Science and Technology, 8(1), 8-13.

[114] Mariano, C. E., & Morales, E. 1999 (June). A Multiple Objective Ant-Q Algorithm for the Design of Water Distribution Irrigation Networks. Tech. rept. HC-9904. Instituto Mexicano de Tecnologa del Agua, Mexico.

[115] Iredi, S., Merkle, D., & Middendorf, M. 2001. Bi-Criterion Optimization with Multi Colony Ant Algorithms. Pages 359-372 of : First International Conference on Evolutionary Multi-criterion Optimization (EMO'01), vol. 1993. Lecture Notes in Computer Science.

[116] Doerner, K., Gutjahr, W. J., Hartl, R. F., Strauss, C., & Stummer, C. 2004. Pareto Ant Colony Optimization : A Metaheuristic Approach to Multiobjective Portfolio Selection. Annals of Operations Research.

[117] Barán, B., & Schaerer, M. 2003. A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows. Pages 97-102 of : Proc. Twenty first IASTED International Conference on Applied Informatics, Insbruck, Austria.

[118] Ulungu, E. L., Teghem, J., Frtemps, P., & Tuyttens, D. 1998. MOSA method : A tool for solving multi-objective combinatorial optimization problems. Tech. rept. Laboratory of Mathematic and Operational Research, Faculté polytechnique de Mons.

[119] Doerner, K., Hartl, R. F., & Teimann, M. 2003. Are COMPE-Tants More Competent for Problem Solving ? The Case of Full Truckload Transportation. Central European Journal of Operations Research, 11(2), 115-141.

[120] Cardoso, P., Jesus, M., & Márquez, A. 2003. MONACO - Multi-Objective Network Optimisation based on ACO. In : In X Encuentros de Geometría Computacional, Seville, Spain.

[121] M.P.Hansen and A.Jaszkiewicz. Evaluating the quality of approximations of non dominated set. Tech.Rep., Institute of Mathematical modeling, Tech.Univ of Denmark, 1998.IMMTech.Rep.IMM-REP-1998-7.

[122] J.D.Knowles and D.W.Corne. On metrics for comparing non-dominated sets. In Congress on Evolutionnary Computation (CEC'02),IEEEPress,pages711–716,2002.

[123] E.Zitzler, L.Thiele, M.Laumanns,C.M Fonseca, and V.G.da Fonseca. Performance Assessment of multiobjective optimizers: ananalysis and review. IEEE Transactions On Evolutionary Computation,7(2):618–630,2003.

[124] Alaya, I., Solnon, C., & Ghédira, K. 2004. Ant algorithm for the multi-dimensional knapsack problem. Pages 63-72 of : Proceedings of International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004).

[125] D.Van Veldhuizen. Multiobjective Evolutionary Algorithms: Classifications, Analyses, and nex Innovations. PhD thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio ,May1999.

[126] M.Basseur, F.Seynhaeve, and E.-G.Talbi. Design of multiobjective evolutionary algorithm: application to the flowshop scheduling problem. In Congress on Evolutionnary Computation (CEC'02), IEEE Press, pages1151–1156, 2002.

[127] E.Zitzler. Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications. PhD thesis, Swiss Federal Institute of Technology(ETH),Zurich, Switzerland, November1999.

[128] Chu, P.C., & Beasley, J.E. 1998. A genetic algorithm for the multidimentional knapsack problem. Journal of heuristics, 4, 63-86.

[129] Alaya, I., Solnon, C., & Ghédira, K. Optimisation par colonies de fourmis pour le problème du sac à dos multidimensionnel. Techniques et Sciences Informatique, 26(3-4).

[130] Dorigo, M. 1992. Optimization, Learning and Natural Algorithms (in Italian). Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

[131] Dorigo, M., & Gambardella, L.M. 1997. Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1), 53-66.

[132] Gambardella, L., Taillard, E., & Dorigo, M. 1999a. Ant Colonies for the Quadratic Assignment Problem. Journal of the Operational Research Society, 50, 167-176.

[133] Bullnheimer, B., Hartl, R.F., & Strauss, C. 1999. An Improved Ant system Algorithm for the Vehicule Routing Problem. Annals of Operations Research, 89, 319-328.

[134] Morrison, R.W, & DeJong, K.A. 2001. Measurement of population diversity. Pages 31-41 of : In 5th International Conference EA 2001, vol.2310 of LNCS. Springer-Verlag.

[135] Dorigo, m., & di caro, g. 1999. The ant colony optimization meta-heuristic. pages 11-32 of : corne, d., dorigo, m., & glover, f. (eds), new ideas in optimization. mcgraw hill, uk.

[136] 3 Dorigo, M., & Stüzle, T. 2004. Ant Colony Optimization. Springer-Verlag.

[137] Alaya, I., Sammoud, O., Hammami, M., & Ghédira, K. 2003. Ant Colony Optimization for the k-Graph Partitioning Problem. In : The 3rd Tunisia-Japan Symposium on Science and Technology, TJASST2003.

[138] Alaya, I., Solnon, C., & Ghédira, K. 2005. Différentes stratégies phéromonales pour le sac a dos multidimensionnel. Pages 151-160 of : Méthodologies et Heuristiques pour l'Optimisation des Systèmes Industriels(MHOSI 2005).

[139] Leguizamon, G., & Michalewicz, Z. 1999. A new version of Ant System for Subset Problem. Pages 1459-1464 of : Proceedings of Congress on Evolutionary computation.

[140] Saha, A. , Maamri R. A New Pheromonal Strategy in Ant Algorithm for Multi-Objective Knapsack Problems. MASAUM Journal of Basic and Applied Sciences (MJBAS) Vol.1 No.1 (August2009).

[141] Solnon, C. 2002. Ants can Solve Constraint Satisfaction Problems. IEEE Transactions on Evolutionary Computation, 6(4), 347-357.

[142] Gambardella, L. M., & Dorigo, M. 1995. Ant-Q : A Reinforcement Learning Approach to the Traveling Salesman Problem. Pages 252-260 of : Proc. Twelfth International Conference on Machine Learning (ML-95).

[143] Zitzler, E., & Thiele, L. 1999. Multiobjective evolutionary algorithms : a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation, 257-271.

[144] Srinivas, N., & Deb, K. 1994. Multiobjective optimization using non dominated sorting in genetic algorithms. Evolutionary Computation, 2, 221-248.

[145] Hajela, P., & Lin, C-Y. 1992. Genetic search strategies in multi-criterion optimal design. Structural Optimization, 99-107.

[146] Horn, J., Nafpliotis, N., & Goldberg, D.E. 1994. A niched pareto genetic algorithm for multiobjective optimization. Pages 82-87 of : Center, IEEE Service (ed), First IEEE Conference on Evolutionary Computation, vol. 1. Piscataway: IEEE World Congress on Computational Computation.

[147] Schaffer, J.D. 1985. Multiple objective optimization with vector evaluated genetic algorithms. Pages 93-100 of : Grefenstette, J.J (ed), ICGA International Conference on Genetic Algorithms. Lecture Notes in Computer Science.

[148] Zitzler, E., Laumanns, M., & Thiele, L. 2001. SPEA2 : Improving the Strength Pareto Evolutionary Algorithm. Pages 12-21 of : In : K. Giannakoglou et al. (Eds.) EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece.

[149] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. 2002. A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II. Pages 182-197 of : IEEE Transactions on Evolutionary Computation, vol. 6 :2.

[150] Fonseca, C.M., & Fleming, P.J. 1993. Genetic algorithms for multi-objective optimization : formulation, discussion and generalization. Pages 416-423 of : The Fifth International Conference on Genetic Algorithms.

[151] Garcia-Martinez, C., Cordon, O., & Herrera, F. 2007. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for bi-criteria tsp. Pages 116-148 of : European Journal of Operational Research.