UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CONCEPTION PAR BLOCS EN UTILISANT L'OPTIMISATION ÉVOLUTIVE : OPTIMISATION PAR ORGANISME MULTICELLULAIRE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

PATRICK FAFALI AGBOKOU

OCTOBRE 2018

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.07-2011). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

AVANT-PROPOS

Je remercie Christine A.E.B., Pierre K.A.A., Amélie L., Henri A., qui par leur soutien moral et financier ont rendu cette étude possible.

Je remercie le gouvernement du Québec, l'UQAM et les professeurs Mounir Boukadoum et Étienne Gagnon, qui, de par leurs conseils, ont poussé ces travaux au-delà des limites que je croyais infranchissables.

Je remercie tous les êtres qui, à un moment ou un autre de mon humble existence, m'ont transmis leur savoir, car ces travaux ne sont pas uniquement les miens, mais ils sont le fruit de l'amalgame de tous ces êtres qui ont fait partie de ma vie, et qui ont fait de moi, ce que je suis.

TABLE DES MATIÈRES

CHAPITRE IV

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Dans cette étude, une nouvelle approche heuristique de recherche et d'optimisation, inspirée de la croissance des organismes multicellulaires (MCOO *Multicellular Organism Optimization*), est proposée afin d'offrir une plus grande vitesse de convergence que deux autres méthodes heuristiques de recherche et optimisation inspirées de la nature, qui utilisent l'intelligence des essaims : la recherche gravitationnelle (GSA) et l'optimisation par essaim de particules (PSO). Pour cela, elle est expérimentalement évaluée par rapport à GSA et PSO. Le niveau d'abstraction qu'offre la méthode proposée nous permet de l'appliquer à un problème réel qui est celui de l'ordonnancement des RSUs (*Road Side Units* : unités de bord de route).

MOTS CLÉS : optimisation, combinatoire, PSO, MCOO, optimisation par organisme multicellulaire, composition de transformations, RSUs, algorithme évolutionnaire, optimisation dynamique, communication inter-véhiculaire,

INTRODUCTION

Les techniques d'optimisation sont classées par le type d'espace de recherche et de la fonction objectif (représentation formelle de l'objectif à atteindre). La programmation linéaire (LP) est la plus simple avec une fonction objectif linéaire. Si les variables d'entrée de LP sont restreintes à des entiers on parle de programmation entière (IP ou ILP). Dans le cas où la fonction objectif n'est pas linéaire, on parle de programmation non linéaire (NLP) (del Valle *et al.*, 2008).

Les problèmes réels ont souvent des fonctions objectif non linéaires. À cause de la difficulté de résolution des problèmes de programmation non linéaire avec des méthodes exactes, de nouvelles techniques ont été développées afin de rendre faisable leur résolution : c'est ainsi qu'est apparue la programmation stochastique.

En outre, la nature a toujours été une source d'inspiration pour solutionner les problèmes (Das et Suganthan, 2011), y compris ceux d'optimisation, car ses processus, malgré leur apparence chaotique, aboutissent souvent à un ordre ou une amélioration. Das et al. disent dans le même ordre d'idée qu'une observation des techniques de la nature a conduit au développement d'un important paradigme d'intelligence computationnelle : les techniques de calcul évolutionnaires pour accomplir des tâches de recherche et d'optimisation complexes.

L'utilisation du paradigme évolutionnaire date des années 50 avec la programmation évolutionnaire (EP) par Lawrence J. Fogel (1961), les stratégies d'évolution (ES) par Rechenberg et Schwefel (1963), puis, plus tard, avec John Henry Holland qui simule le processus darwinien avec son algorithme génétique (GA)(1965). Une tendance particulière voit le jour dans les années 90 avec la programmation génétique (GP) (Heudin, 2008).

De nos jours, ces algorithmes inspirés de la nature, sont nombreux. On y retrouve les GAs et leurs variantes, l'évolution différentielle (DE *Differential Evolution*), le recuit simulé (Kirkpatrick *et al.*, 1983), (SA *Simulated Annealing*), inspiré de la métallurgie. Plus récemment, une famille d'algorithmes, aussi inspirés de la nature, qui a gagné en popularité est celle des algorithmes qui simulent l'intelligence des essaims. Dans cette famille, se trouvent l'optimisation par colonie de fourmis (ACO) (Dorigo *et al.*, 1996), qui reproduit le comportement des fourmis à la recherche de nourriture, la recherche gravitationnelle (GSA), qui simule l'attraction gravitationnelle des corps célestes (planètes, étoiles, ...), l'optimisation par essaims de particules (PSO), qui simule le comportement d'une volée d'oiseaux (Kennedy et Eberhart, 1995a), l'algorithme des abeilles (ABC *Algorithm Bee Colony*), l'optimisation par bactéries à la recherche de nourriture (BFO), le système immunitaire artificiel de Farmer et Al. (Farmer *et al.*, 1986).
Ces algorithmes ont été appliqués avec un certain succès à des problèmes de la classe NP-difficile, surtout GSA et le populaire PSO. Ces algorithmes inspirés de la nature compétitionnent pour la rapidité. C'est ainsi que la recherche gravitationnelle a été conçue afin de proposer une vitesse de convergence meilleure à celle de PSO (Rashedi *et al.*, 2009). Nous voyons plus en détails le mécanisme qui permet d'accomplir cela dans le chapitre suivant. C'est dans ce même ordre d'idée que nous proposons un algorithme que nous nommons Optimisation par Organisme Multi-cellulaire ou MCOO (*Multi-Cellular Organism Optimization*) et qui a pour but d'offrir une vitesse de convergence plus grande que GSA.

Le document est subdivisé comme suit. Le premier chapitre étudie l'applicabilité de GSA et PSO, par rapport à des problèmes dynamiques. C'est-à-dire des problèmes dont l'espace des variables change dans le temps. Certains algorithmes dignes de mention par rapport aux problèmes dynamiques sont aussi survolés brièvement afin d'avoir une vision plus large des possibilités offertes en termes de

techniques d'optimisation. Cependant, ils ne sont pas comparés à notre proposition, car leurs vitesses de convergence sont inférieures à celles de GSA et PSO, comme montré par les auteurs de GSA.

Le deuxième chapitre présente notre proposition, MCOO, et montre comment elle peut être appliquée à l'optimisation en espace continu et à la recherche combinatoire, ses limites et ses variantes possibles.

Le troisième chapitre vérifie d'abord que MCOO s'extirpe des optima locaux et converge vers l'optimum (ou les optima) globaux, et en plus, donne les résultats attendus, c'est-à-dire, une vitesse de convergence plus grande que GSA et PSO. Rappelons qu'un optimum local est une solution optimale dans le voisinage d'une ou plusieurs solutions candidates.

Le quatrième chapitre montre que MCOO peut être effectivement appliqué à des problèmes réels, car leur niveau de complexité, en termes de variables, objectifs et contraintes, est souvent plus grande que celui des cas de laboratoire. Pour cela, nous choisissons un problème en vogue qui est l'ordonnancement des RSUs. Ce chapitre décrit plus en détails ce problème particulier.

CHAPITRE I

ÉTAT DE L'ART ET LIMITES

Russel et Norvig définissent les techniques heuristiques comme étant des techniques qui recherchent des solutions optimales ou presque optimales à un coût de calcul raisonnable sans pouvoir garantir la faisabilité ou l'optimalité. (Russell et Norvig, 1995).

Une technique heuristique est un algorithme. Dans le cas de l'optimisation, elle peut être conçue comme une procédure d'identification d'une solution optimale ou presque optimale à un problème. Un problème est un ensemble d'objectifs et de contraintes définis sur un domaine. Un problème peut avoir ou non une solution. Un optimum global est la meilleure solution possible au problème, en tenant compte de tous les éléments du domaine. Un optimum local est la meilleure solution possible au problème, en tenant compte d'un sous-ensemble du domaine.

Un problème d'optimisation sur espace discret, ou problème d'optimisation combinatoire, est un problème d'optimisation dont l'espace des variables est constitué de points isolés. On retrouve dans cette catégorie les problèmes à variables catégoriques ou booléennes.

Un problème d'optimisation en horizon fini est un problème d'optimisation dont l'espace des variables est statique. Un problème d'optimisation en horizon infini est un problème d'optimisation dont l'espace des variables est dynamique (variations possibles sont infinies). On y retrouve bien entendu les problèmes d'optimisation

dynamique tel que celui de l'agent rationnel.
Il est décrit, dans la suite, des méthodes heuristiques du moment et les limites dont elles souffrent par rapport aux problèmes dynamiques, ce qui nous permet de justifier la contribution MCOO.

1.1 NSGA-II

Nous utilisons NSGA-II (*Nondominated Sorting Genetic Algorithm*) pour illustrer les GAs traditionnels (approches heuristiques de recherche et d'optimisation inspirées de la théorie de l'évolution), car c'en est une variation qui se focalise sur la recherche multi-objectif (recherche de solution répondant simultanément à plusieurs critères, quelques fois contradictoires).
Sa particularité est le calcul de la dominance, lors de l'opération de sélection, selon le critère de Pareto (Deb *et al.*, 2002). C'est-à-dire, que l'état optimal global est l'état qui ne peut être amélioré sans la détérioration de l'état de l'un des individus du groupe. La solution candidate est représentée par un chromosome qui évolue de manière aléatoire (Deb *et al.*, 2002). Les opérations les plus courantes sont la sélection, la mutation et le croisement.
Depuis son ajout par Deb et al. NSGA-II a été utilisé sur des problèmes très variés comme la planification des centres de livraisons par Azadeh et al. (Azadeh *et al.*, 2016). NSGA-II fonctionne comme suit :

Algorithm 1: NSGA-II

```
1 Initialiser la population générale de chromosomes;
2 Initialiser la population élite de chromosome à l'ensemble vide;
3 while itération maximale ou les critères d'erreur minimale ne sont pas
    atteints do
4     for chaque chromosome de la population do
5         Appliquer Mutation/Croisement;
6         Calculer la qualité du chromosome (fitness);
7     Appliquer Sélection (Pareto) : les meilleurs de la population générale
      sont copiés dans la population élite;
8     Vider la population générale et y mettre les chromosomes de la
      population élite;
```

Cette représentation chromosomique porte en elle un frein qui est la taille fixe des gènes et/ou du chromosome.

Prenons l'exemple d'une simple recherche d'arcs afin de construire un graphe. Dû à cette représentation chromosomique (donc uni-chemin) de la solution, les graphes multi-chemins (représentations multidimensionnelles) sont difficiles (pas impossibles) à représenter, à cause de deux problèmes :

- La taille fixe des gènes demande de connaître a priori tous les arcs possibles afin de pouvoir les encoder en gènes pour l'ensemble des variables. Ce nombre d'arcs explose avec l'augmentation du nombre de sommets.
- La taille fixe du chromosome impose une limite dure au nombre maximal d'arcs dans le graphe solution.

Le nombre de sommets et d'arcs ne peut être connu a priori avec les graphes dynamiques, ce problème particulier est donc impossible à représenter de manière chromosomique. NSGA II n'est donc pas adéquat pour les problèmes dynamiques

(comme celui de la composition de transformations *cf.* Annexe A).
Une population de solutions est conservée à chaque génération et les candidats les moins performants sont rejetés par l'opération de sélection. Compte tenu de la nature des problèmes dynamiques qui exige que l'espace des variables varie, cette destruction de solutions candidates, donc de variables par NSGA II est contre-productive pour les problèmes combinatoires où les performances d'une partie de la solution et de la solution elle-même sont indépendantes (une solution candidate non performante rejetée peut être la partie d'une solution idéalement performante).
Pour illustrer cet effet d'indépendance des solutions, supposons que nous recherchons la composition de portes logiques qui reçoit en entrée de données binaires en série et les transforme en une autre série. Cette série représente les états d'une ligne de transmission dans le temps. De manière simpliste, les entrées et sorties de cette composition sont représentées par des listes de booléens. Une telle solution candidate peut produire une liste de booléens qui correspond à 80% à liste de booléens de la solution idéale, et la liste de booléens de sa composition avec une autre solution candidate peut correspondre juste à 50% de la solution idéale.
L'addition de gène est possible afin de pouvoir tacler les problèmes dynamiques. Cependant, cette addition de gène porte en elle un frein, car le nouveau gène doit avoir la même taille que les gènes existants. Augmenter la taille des gènes implique de redéfinir l'encodage des solutions candidates.
Le parcours itératif des solutions, requis pour la création de nouvelles, ralentit l'algorithme.

1.2 Optimisation par essaim de particules

L'optimisation par essaim de particules PSO (*Particle Swarm Optimization*) est une approche heuristique de recherche et d'optimisation qui repose sur le compor-

tement des essaims. La solution candidate est une particule et les transformations appliquées représentent ses déplacements (vélocité) vers la solution ultime (Lim et Haron, 2013).

Depuis la conception de PSO, en 1995, par Kennedy et Eberhart (Kennedy et Eberhart, 1995b), de nombreux chercheurs l'ont utilisée afin d'aborder des problèmes d'optimisation non linéaires, comme celui de l'optimisation des circuits, gros comme les systèmes de distributions de courant (Panda *et al.*, 2013; Al-Saedi *et al.*, 2017), aux petits comme des composants de circuits. Patel et al. utilisent PSO afin de générer des *netlists* pour le design d'un amplificateur opérationnel (Patel et Thakker, 2016). Leur méthode de conception est aussi du type "simulateur-dans-la-boucle". Ils ont contourné la limite inhérente de PSO par plusieurs exécutions complètes avec un nombre fixe de composants.

La position et la vélocité des particules sont respectivement calculées comme suit :

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{1.1}$$

$$v_i^d(t+1) = w(t)v_i^d(t) + c_1 r_{i1}\left(pbest_i^d - x_i^d(t)\right) + c_2 r_{i2}\left(gbest^d - x_i^d(t)\right), \tag{1.2}$$

où r_1 et r_2 sont des nombres aléatoires appartenant à l'intervalle $[0,1]$, c_1 et c_2 sont des constantes positives. w est le poids d'inertie. $X_i = (x_i^1, x_i^2, \ldots, x_i^n)$ et $V_i = (v_i^1, v_i^2, \ldots, v_i^n)$ la position et la vélocité de la i^{eme} particule. $pbest_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^n)$ est la meilleure position précédente de la particule i, et $gbest = (gbest^1, gbest^2, \ldots, gbest^n)$ la meilleure position précédente globale. Voici comment fonctionne PSO :

Algorithm 2: PSO

```
1  for chaque particule do
2      Initialiser la particule (position et vélocité aléatoire)
3  do
4      for chaque particule do
5          Calculer la fitness;
6          if la fitness est meilleure que la meilleure fitness then
7              Définir la fitness actuelle comme nouvelle meilleure fitness;
8      for chaque particule do
9          Trouver dans le voisinage des particules, la particule avec la
           meilleure fitness;
10         Calculer la vélocité de la particule en fonction de l'équation de la
           vélocité;
11         Appliquer les contraintes de la vitesse;
12         Mettre à jour la position de la particule en fonction de l'équation de
           position;
13         Appliquer les contraintes de position;
14 while l'itération maximale ou les critères d'erreur minimale ne sont pas
   atteints;
```

PSO a été développée pour sa convergence rapide (Aote *et al.*, 2013), mais comme le montre l'algorithme Algorithm 2 ci-dessus, PSO, nativement, ne conserve pas les modifications des positions et vélocités des particules à travers les générations. Cette représentation fixe de la solution, qui est la position de la particule dans l'espace de recherche, contient la limite inhérente que l'espace des variables doit avoir un nombre de dimensions fixe, qui est déterminé de manière statique. Même si l'ajout de coordonnées est une opération envisageable, PSO n'est pas nativement

adéquat pour les problèmes dynamiques, pour les mêmes raisons que NSGA-II : la dimensionnalité fixe de la représentation de la solution et la destruction de solutions candidates.

Les modifications apportées aux solutions reposent entièrement sur le calcul du déplacement de la particule et de la meilleure de ses voisines. Ce qui fait que PSO ne peut opérer nativement qu'avec des variables continues. L'utilisation des variables discrètes exige une conversion vers un domaine continu (variable booléenne rendue continue entre 0 et 1 fait perdre de la précision au calcul de fitness, ne permet pas de se rappeler efficacement des cas déjà évalués par exemple) qui alourdit et ralentit la recherche.

Aussi, la création de nouvelles solutions, implique une connaissance de plusieurs solutions voisines (donc un parcours itératif des solutions).

1.3 Recherche gravitationnelle

La recherche gravitationnelle GSA (*Gravitational Search Algorithm*), proposée par Rashedi et al.(Rashedi *et al.*, 2009), est une approche heuristique de recherche et d'optimisation qui s'inspire de l'attraction gravitationnelle des corps célestes (planètes, étoiles,...). L'élément structurel est l'agent qui représente un corps céleste. Le principe, de manière simplifiée est : plus la solution candidate (la position) est bonne, plus sa masse gravitationnelle augmente, donc, plus sa force d'attraction augmente, et plus sa masse inertielle augmente, donc, moins vite elle se déplace. GSA a été utilisée dans des travaux de gestion de lignes de production comme ceux de Yaping et al. (Ren *et al.*, 2017). GSA fonctionne comme suit :

Algorithm 3: GSA

```
1 Initialiser la population initiale;
2 do
3     for chaque agent do
4         Calculer la qualité;
5         Mettre à jour la constante gravitationnelle, la meilleure et la pire
          fitness;
6         Calculer la masse gravitationnelle et la force qui agit sur l'agent;
7         Mettre à jour la vitesse et la position de l'agent;
8 while l'itération maximale ou les critères d'erreur minimale ne sont pas
    atteints;
```

Rashedi et al. décrivent ainsi leur proposition. En considérant un système de n agents X :

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) \quad \text{for } i = 1,2,\ldots,N, \tag{1.3}$$

où x_i^d est la position du i^{eme} agent dans la d^{ieme} dimension.

À la date t, la force qui agit sur l'agent i par l'agent j est calculée comme suit :

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)), \tag{1.4}$$

où M_{aj} est la masse gravitationnelle active reliée à l'agent j, M_{pi} est la masse gravitationnelle passive reliée à l'agent i, $G(t)$ la constante gravitationnelle à l'instant t, ϵ une constante et $R_{ij}(t)$ est la distance euclidienne entre i et j.

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2 \tag{1.5}$$

La force totale qui s'applique à l'agent i dans la dimension d est supposée stochastique et est calculée comme suit.

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{1.6}$$

où $rand_j$ est un nombre aléatoire appartenant à $[0, 1]$.

L'accélération de l'agent i à la date t dans la dimension d est :

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{1.7}$$

où M_{ij} est la masse inertielle du i^{eme} agent.

La vélocité et la position sont ainsi calculées :

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t), \tag{1.8}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{1.9}$$

La constante gravitationnelle est réduite en fonction du temps selon :

$$G(t) = G(G_0,t). \tag{1.10}$$

où G_0 est une valeur initiale choisie. La masse gravitationnelle et la masse inertielle sont calculées comme suit :

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{1.11}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{1.12}$$

où $fit_i(t)$ est la fitness de l'agent i à la date t, avec :

$$best(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t), \tag{1.13}$$

$$worst(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t). \tag{1.14}$$

La convergence de GSA est meilleure que celle de PSO à cause du principe d'attraction qui augmente de manière inversement proportionnelle avec la distance entre la solution candidate et la solution recherchée.

La représentation de la solution en GSA, qui est un agent ayant une masse est aussi rigide que celle de PSO. Elle souffre des mêmes limites que PSO, telles que le

nombre fixe de dimensions de l'espace des variables, qui est déterminé de manière statique. Les problèmes dynamiques sont encore nativement difficiles à représenter (à cause de leur dimensionnalité dynamique) avec cet algorithme pour les mêmes raisons que PSO.

Les modifications apportées aux solutions reposent entièrement sur le calcul de la vitesse et de la position des points. Ce qui fait que GSA ne peut opérer aussi qu'avec des variables continues. Le même problème des séries de booléens qui se pose avec PSO, se pose avec GSA.

Aussi, le calcul de la force qui agit sur un agent implique une connaissance de tous les autres agents (donc un parcours itératif des solutions).

1.4 Programmation génétique

La programmation génétique GP (*Genetic Programming*) est une méthode heuristique de recherche et optimisation, inventée par Koza et aussi inspirée de la sélection naturelle, afin de créer des programmes en assemblant des parties de programmes. La représentation de la solution est un programme construit à partir du parcours d'un arbre (Banzhaf *et al.*, 1998). C'est cet algorithme que Koza et al. utilisent pour générer des circuits où les éléments fonctionnels représentent des composants analogiques (KOZA *et al.*, 2004).

Bien que cette représentation de solutions permette nativement des tailles variables de chromosome (des solutions de taille non prédéfinis), une limite inhérente à la représentation en arbre est la difficulté de représenter des références cycliques dans la solution (un descendant ne peut être le parent direct ou indirect d'un ou plusieurs de ses ascendants à moins d'être dupliqué dans une autre branche.

Une ambiguïté est introduite, par cette duplication, quand la connaissance préalable du domaine est manquante : la similitude représente-t-elle l'instance elle-même ou une instance similaire séparée ? Cette distinction est très importante à

faire, surtout dans le cas des programmes ou parties de programmes dont l'entrée modifie l'état interne et la sortie dépend de ce dernier. Dans ces cas, l'instance et sa copie n'ont pas la même sortie, si soumises à des entrées différentes.

Cette technique de représentation rend encore moins triviale la représentation des solutions structurées en graphes multi-chemins (graphes caractérisé par des sommets pouvant être liés par plusieurs chemins *cf.* Annexe A). Des représentations cartésiennes ont été proposées afin de représenter des graphes multi-chemins (Wilson et Banzhaf, 2008). Cette représentation consiste en l'utilisation des matrices au lieu d'arbres. Cependant, ces variantes contiennent des limitations dimensionnelles inhérentes à la représentation matricielle de la solution.

Les modifications apportées par la GP aux solutions candidates peuvent être la mutation d'un nœud ou le croisement par la permutation de sous-arbres. Dans le cas de la représentation cartésienne, le croisement sur des matrices, n'est pas intuitif, donc seule la mutation est utilisée.

En outre, il existe une limite au degré de composition *cf.* Annexe A, due à la représentation matricielle. Un paramètre est introduit afin de spécifier une limite au nombre de niveaux à remonter. Ceci est nécessaire pour connecter un nœud d'un certain niveau à celui d'un niveau précédent. Cette limite au nombre de niveaux est déterminée de manière statique.

Pour ces raisons, GP n'est donc pas adéquat pour le problème de la composition de transformations.

L'indépendance des solutions permet la parallélisation native de cet algorithme.

1.5 Évolution différentielle

l'évolution différentielle DE (*Differential Evolution*) est une méthode heuristique de recherche et d'optimisation conçue spécialement pour la découverte de gradient. DE opère sur des espaces vectoriels en combinant deux vecteurs pour en produire

un autre. Rappelons que le gradient est la forme généralisée aux vecteurs de la notion de dérivée et que la découverte de gradient consiste à déterminer la direction dans laquelle l'on se déplace, étant donné un vecteur de départ et un vecteur d'arrivée. DE fonctionne ainsi :

Algorithm 4: GSA

```
1  CR est la probabilité de croisement et est entre 0 et 1 inclusivement;
2  F est le poids differentiel et est entre 0 et 2 inclusivement;
3  Initialiser la population de vecteurs x;
4  do
5      for chaque vecteur x de la population do
6          Sélectionner trois vecteurs a,b et c;
7          Sélectionner un indice aléatoire R inférieur ou égal à la
           dimensionnalité des vecteurs;
8          for chaque position i de l'indice, inférieur ou égal à la
           dimensionnalité des vecteurs do
9              Choisir un nombre uniformément distribué ri entre 0 et 1;
10             if ri < CR ou i = R then
11                 yi = ai + F * (bi - ci);
12             else
13                 yi = xi;
14             if fitness de y > fitness de x then
15                 Remplacer x par y dans la population;
16 while l'itération maximale ou les critères d'erreur minimale ne sont pas
   atteints;
```

La représentation de base de la solution est un vecteur de nombre réels (Lim et Haron, 2013). Cette représentation limite la dimensionnalité de l'espace de recherche

à celle du vecteur et rend difficile la représentation des graphes dynamiques, à l'image de NSGA-II, PSO et GSA.

Dans l'algorithme original, deux solutions candidates sont sélectionnées et combinées de façon aléatoire afin d'obtenir une solution candidate plus performante. Les solutions les moins performantes sont rejetées de la population. La solution ne croît pas dans l'espace.

La sélection aléatoire de vecteurs pour la création de nouveaux, implique une connaissance de tous les vecteurs de la population (donc un parcours itératif des solutions).

DE est donc aussi inadéquat pour les problèmes dynamiques, pour les mêmes raisons que NSGA-II, PSO et GSA.

1.6 Recuit simulé

Le recuit simulé SA (*Simulated Annealing*) est une méthode heuristique de recherche et d'optimisation inspirée du processus de recristallisation des métaux par des échauffements et des refroidissements contrôlés. Elle fonctionne ainsi :

Algorithm 5: SA

```
1  P est une fonction de probabilité dépendant de la variation d'énergie et de
   la température;
2  Choisir la première solution s0 candidate comme solution s;
3  Calculer l'énergie e de cette solution avec E(s);
4  Mettre la température k à 0;
5  while température k < température maximale kmax et l'énergie e > emax
   do
6      Selectionner un voisin de s dans sn;
7      Calculer l'énergie en du voisin avec E(sn);
8      if en < e ou un nombre aléatoire quelconque inférieur à P(en - e,
       temperature(k/kmax)) then
9          Remplacer s par sn et e par en;
10     Incrementer k;
```

La solution est représentée par la structure du métal qui est recuit (Lutton et Bonomi, 1986). La définition exacte de la représentation computationnelle de la solution est laissée au domaine. La seule exigence est que le calcul d'une température soit possible avec cette représentation. Cette représentation de par son niveau d'abstraction, n'impose pas de limite sur le type des variables. Ceci est un avantage par rapport à PSO et GSA. Toutefois, une mauvaise représentation du domaine peut ralentir la recherche. Le problème de la meilleure représentation du problème est ainsi soulevé.

La principale modification utilisée pendant la recherche du niveau d'énergie est la perturbation. Dépendamment de la température, une nouvelle solution peut être acceptée ou non.

La recherche est ralentie par le fait qu'une seule solution est conservée à chaque

génération (Bailey *et al.*, 1997).
Les nouvelles solutions candidates sont créées à partir de solutions candidates voisines, ce qui implique une connaissance de toutes les solutions de la population (donc un parcours itératif des solutions).Pour ces raisons, SA est donc aussi inadéquat pour les problèmes dynamiques que NSGA-II.

Dans le chapitre suivant, nous présentons la méthode MCOO, ses forces et ses faiblesses pour les deux principaux types de problèmes d'optimisation et nous montrons pourquoi il est plus adéquat pour les problèmes dynamiques.

1.7 Problématique

Ces algorithmes possèdent des limites rigides par rapport aux problèmes dynamiques, à l'exception de GP, qui souffre, soit de la confusion entre les instances avec la représentation en arbre, soit de la limite du nombre de niveaux avec la représentation cartésienne. Est-il possible de leur proposer une alternative pour ce type de problèmes, et en plus une alternative plus efficace en termes de vitesse de convergence ?

1.8 Méthodologie

Afin de répondre à cette question, nous évaluons, dans un premier temps, notre proposition, MCOO, par rapport à des fonctions de benchmarking (Ackley, Rastrigin, ...), qui montrent clairement, si notre proposition peut en plus de s'extirper des optima locaux de ces fonctions, converge plus vite vers leurs optima globaux. Puis, nous évaluons MCOO par rapport à des problèmes de recherche combinatoire, afin de s'assurer qu'elle peut être nativement appliquée à ce type de problèmes. Ensuite, MCOO est validée avec un problème de plus grande envergure, l'ordonnancement des RSUs.

CHAPITRE II

OPTIMISATION PAR ORGANISME MULTICELLULAIRE : LIMITES ET VARIANTES

Dans un premier temps, nous présentons la méthode (MCOO : *Multicellular Organism Optimization*), ses limites et quelques variantes. Ensuite, nous définissons, de manière théorique, comment l'appliquer à la recherche sur un espace continu et à la recherche combinatoire.

2.1 Présentation de MCOO

La méthode proposée est une approche de recherche et d'optimisation heuristique inspirée de la croissance des organismes multicellulaires. Afin de résoudre des problèmes d'optimisation en horizon "infini" (problèmes dont l'espace des variables est infini) [1], elle repose sur le principe des suites convergentes et le principe d'optimalité de Bellman (Bellman, 1957). L'idée générale de l'approche de recherche et d'optimisation proposée est de progresser, de manière récursive, vers l'optimum. Ceci est accompli, en se reposant sur la meilleure performance de l'ensemble des solutions, tout en ayant un certain degré de tolérance à l'erreur, une méthode de correction de cette erreur, et en gardant une indépendance entre les points cher-

1. En considérant les limites choisies du problème rencontré

cheurs (les informatrices).
MCOO n'a pas pour but de suivre le mouvement de l'optimum global (localiser l'optimum à chaque étape de la modification de l'espace des variables) tel que souvent requis en optimisation dynamique (Meyer-Nieberg et Beyer, 2012), mais de s'en rapprocher de plus en plus avec chaque modification de l'espace des variables. MCOO base le critère d'arrêt de la recherche sur les limites choisies du problème et les limites des ressources computationnelles disponibles.

2.1.1 Analogie avec le mode d'opération de la nature

- Une solution est une cellule.
- L'ensemble des solutions candidates est un organisme multicellulaire.
- Une variable du domaine est une aptitude innée de l'organisme.

La cellule est l'une unité structurale générique de MCOO. C'est un assemblage particulier (structure topologique : représentation spatiale) de forme quelconque (matrice, graphe bipartite...), constitué d'une ou plusieurs aptitudes innées de l'organisme. Elle est capable de représenter une solution ou une partie de solution à un problème donné. Sa représentation, est donc par définition, liée au problème. Elle peut être une cellule souche (une variable d'entrée initiale) ou une cellule spécialisée. La croissance de l'organisme s'effectue par l'addition de cellules spécialisées. Une cellule spécialisée est créée par l'amélioration des aptitudes de sa cellule parente en ajoutant, ou retirant des aptitudes innées (variables du domaine) de l'organisme (parallélisation possible, à cause da la nature récursive de la création des nouvelles cellules, une cellule ne connaissant que son parent).
L'objectif ultime de l'organisme, durant sa vie, est de créer une cellule qui s'acquitte (parfaitement ou le mieux possible) d'une tâche spécifique.
Un organisme dont les cellules ne peuvent s'acquitter de la tâche spécifique (résoudre un problème spécifique) doit croître en ajoutant des cellules encore plus

spécialisées. La croissance s'arrête quand une cellule spécialisée s'acquitte parfaitement de la tâche spécifique, ou quand les conditions d'arrêt personnalisées telles que l'âge de l'organisme ou la précision de la solution candidate s'avèrent.

2.1.2 Comment MCOO fonctionne

L'objectif premier de la méthode proposée est de tacler des problèmes dynamiques, par la conservations de parties ou portions de solution durant le processus de recherche. Un exemple est de rechercher ou d'optimiser des structures topologiques dont on ne connaît pas a priori le nombre d'arcs et de sommets. Elle a donc pour but de rechercher des solutions ou d'optimiser dans un espace dont le nombre de dimensions n'est pas connu d'avance. Elle doit, pour cela, découvrir le nombre de dimensions de l'espace de recherche, durant le processus de recherche.
La représentation abstraite de la solution qu'est la cellule suggère simplement que sa spécialisation doit lui permettre une exploration non restreinte de l'espace de recherche. Ainsi, cette abstraction de représentation permet, théoriquement, d'appliquer MCOO à tous les types de problèmes de recherche et d'optimisation, car sa définition concrète (computationnelle) est spécifique au problème. La méthode proposée est donc théoriquement, nativement applicable à des problèmes dont la solution peut être une transformation de variables non quantifiables, soient des qualificateurs (variables nominales ou booléens). Le recuit simulé (Simulated annealing SA) adopte le même principe d'abstraction pour la représentation de la solution afin d'être plus générique.
Afin de pouvoir sortir des optima locaux, MCOO permet une tolérance à la mauvaise qualité (fitness) des solutions candidates, durant l'évolution. MCOO permet donc l'acceptation de solutions sous-optimales à la génération t, dont la spécialisation peut générer des solutions optimales à la génération $t+1$. Par exemple, elle permet de tacler des problèmes d'optimisation dynamique où la solution optimale

à la date t peut être dominée à la date $t + 1$.

En outre, dû à l'indépendance des solutions candidates, MCOO est parallélisable. Il est à noter que la méthode proposée conserve les solutions précédemment trouvées durant la phase de croissance de l'organisme.

Le principe général de MCOO est de combiner à la fois l'exploration et l'exploitation, en créant de nouvelles cellules basées sur les anciennes, avec un espace de variabilité (donc d'exploration) qui se réduit avec l'âge de la cellule. L'âge, lui-même, augmentant avec le nombre de spécialisations subies.

MCOO se distingue de PSO et GSA en cela que les informatrices (points chercheurs) ne sont pas "déplacées" comme dans PSO et GSA (calculs lourds de position et de vélocité), mais générées dans le voisinage des meilleurs points. C'est cette génération spontanée qui lui confère la capacité de découvrir l'espace de recherche lui-même au fur et à mesure que la recherche progresse, car une cellule peut être créée dans une dimension non définie au départ.

L'algorithme Algorithm 7 décrit son fonctionnement de manière formelle, cependant, voici une description sommaire des étapes.

Algorithm 6: MCOO - principe général : fonction optimiser - entrées et sorties

Input: $maxAge$ âge de l'organisme

Input: $specializationRate$ pourcentage de cellules à spécialiser

Input: $growthTolerance$ tolérance à la mauvaise qualité (fitness) d'une nouvelle cellule

Input: $explorationFactor$ nombre de nouvelles cellules à créer

Input: $spreadFactor$ espace de variabilité des nouvelles cellules

Input: $spreadFactorDecay$ coefficient de réduction de l'espace de variabilité

Input: fo fonction d'évaluation de l'adéquation des cellules/fonction objectif

Output: Meilleure cellule

Algorithm 7: MCOO - principe général : fonction optimiser

```
1  function optimiser()
2      qualiteOrganisme ← −1 ;                    // ou le pire cas estimé
3      cellulesSouche ← crercellulessouches ; // Initialiser: vérifier si
        les cellules souches peuvent exécuter la tâche
4      for cellule ∈ (cellulesSouche) do
5          organisme.cellules ← insertionordonneparqualitnouvelleCellule;
6      age ← 0;
7      while age < maxage ou condition d'arrêt non respectée do
8          cellule ← slectionnerparqualitspecializationRate;
9          nouvelleCellules ← spcialisercellule, explorationFactor;
10         for nouvelleCellule ∈ (nouvelleCellules) do
11             if fo(newCell)/fitnessOrganism > (1 − growthTolerance) then
                   ;        // maximisation de l'adéquation (qualité pour
                    fonction objectif+/-contrainte)
12                 organisme.cellules ← insertionordonnepar fitnessnewCell;
13                 if fo(newCell) > qualiteOrganisme then
14                     qualiteOrganisme ← fo(nouvelleCellule);
15         cellule.spreadFactor ← cellule.spreadFactor × spreadFactorDecay;
16         cellule.age ← cellule.age + 1;
17         organisme.age ← organisme.age + 1;
18     return organisme.cellules[tte]
```

La recherche commence avec des cellules possédant une aptitude innée (variable du domaine, élémentaire, de préférence indivisible) qui sont les cellules souches. Ces aptitudes innées doivent être composables (ou combinables). Ces variables élémentaires sont initialisées arbitrairement, s'il n'y en a pas, fig 2.1.

Algorithm 8: MCOO - principe général (cellule générique)

Input: *rand* un générateur aléatoire uniforme de nombres flottants

Input: *explorationFactor* nombre de nouvelles cellules à créer

```
1 function spécialiser(vieilleCellule, explorationFactor)
2     nouvellesCellules ← ∅;
3     k ← explorationFactor;
4     while k > 0 do
5         nouvelleCellule ←
            Creerdansvoisinage(vieilleCellule, spreadFactor, rand);
6         nouvellesCellules ← nouvellesCellules.insrer(nouvelleCellule);
7         k ← k − 1;
8     return nouvelleCellule
```

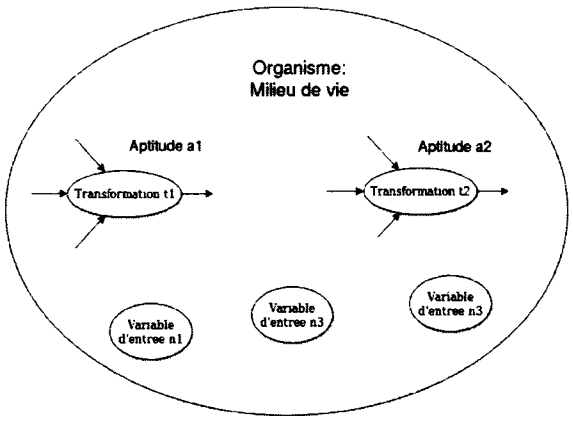


Figure 2.1: Organisme - Age 0

Chaque cellule calcule sa fitness avec la fonction objectif du problème (avec ou sans une fonction de contrainte) et la compare à la fitness de l'organisme (qui est aussi la meilleure fitness), fig 2.3.

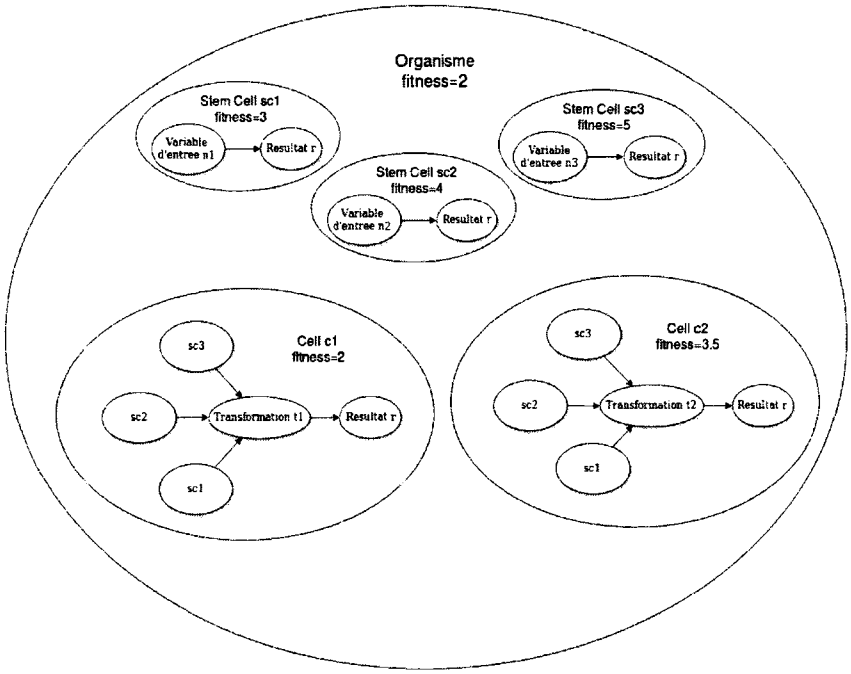


Figure 2.2: Organisme - Age 1

Si la fitness de la cellule dépasse celle de l'organisme, la fitness de l'organisme est mise à jour et la cellule marquée (ou insérée à la tête).

Si la fitness de la cellule se trouve seulement au-dessus du seuil de tolérance à la croissance (marge d'erreur de fitness), elle est simplement marquée (ou insérée à la position relative à sa fitness).

Si la fitness de la cellule se trouve en deçà du seuil de tolérance à la croissance (marge d'erreur de fitness), elle peut être conservée ou détruite. Le comportement à choisir dépend du problème.

De chaque cellule est générée une ou plusieurs cellules filles, avec une augmentation des aptitudes, dont la variation par rapport à leur cellule parente est déterminée par un facteur de variabilité et conditionnée par l'âge de la cellule mère (ajout d'une aptitude aléatoirement sélectionnée), fig 2.3. L'âge de la cellule est incré-

menté.

L'âge de l'organisme est incrémenté.

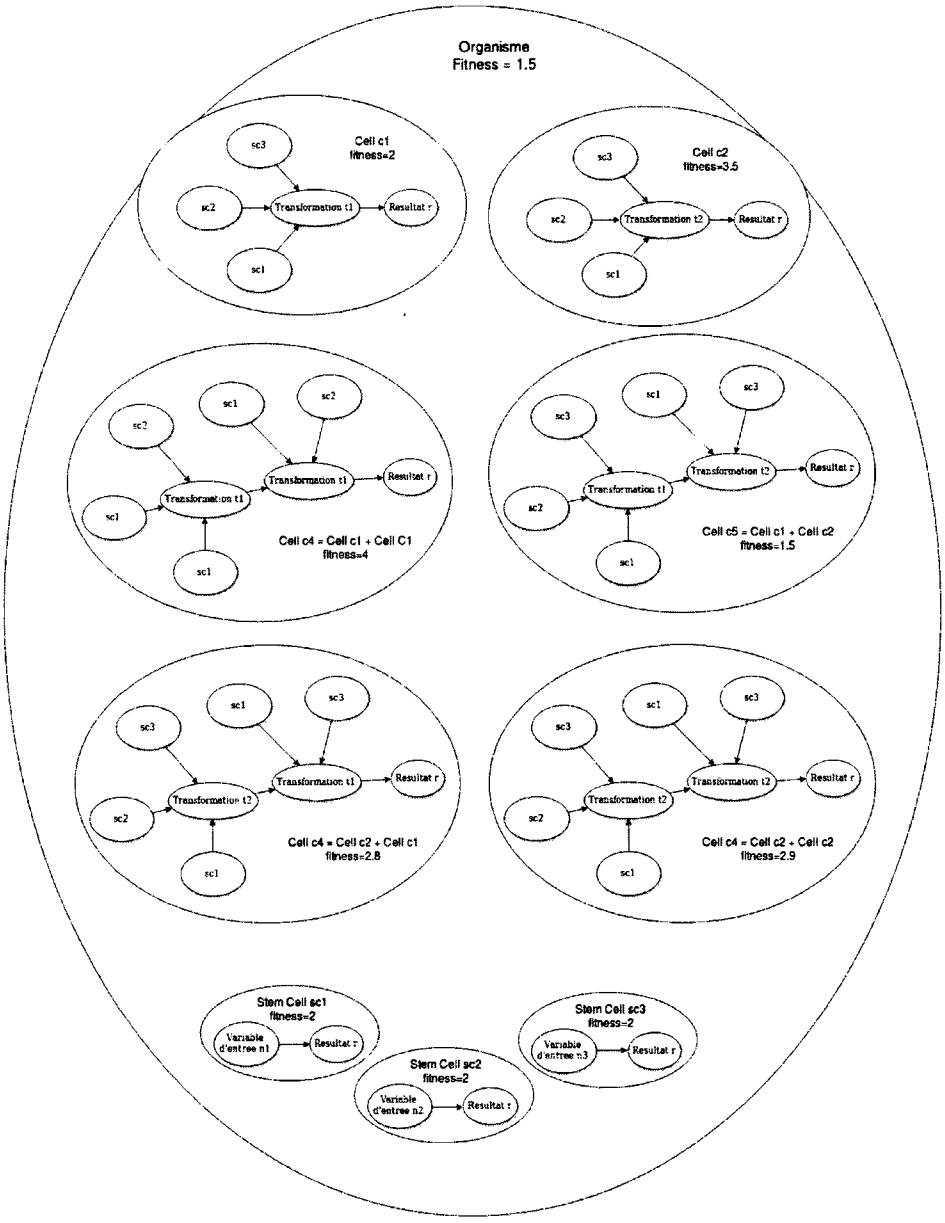


Figure 2.3: Organisme - Age 2

Et le processus recommence tant que l'âge maximal de l'organisme n'est pas atteint

ou que la solution n'est pas trouvée ou n'évolue plus pour un certain nombre d'itérations.

Dans le cas de l'optimisation en espace continu, les cellules moins performantes peuvent être abandonnées, car celles dominantes sont suffisantes pour poursuivre la progression vers l'optimum. Ceci est dû à la relation d'ordre qui lie les éléments du continuum.

Dans le cas de l'optimisation combinatoire, l'isolement (donc, la possible absence de relation d'ordre ou indépendance) des variables de l'espace de recherche ne permet pas de prédire le comportement de la fonction objectif au voisinage d'un point, ce qui rend toutes les cellules importantes pour la poursuite de la progression vers l'optimum.

2.1.3 Comment MCOO s'appuie sur le principe d'optimalité de Bellman

Iwamoto dans son théorème Maximax (Iwamoto, 1985), qui est une reformulation du lemme de Mitten (Mitten, 1964) montre que le principe d'optimalité de Bellman peut s'exprimer de la manière suivante. Si f, une fonction de deux variables x et y, est décomposable suivant :

$$f(x, y) = f_1(x, f_2(y)) \tag{2.1}$$

alors :

$$OPT_{x,y}\{f(x, y)\} = OPT_x\{f_1(x, OPT_y\{f_2(y)\})\} \tag{2.2}$$

où $OPT_x\{f(x)\}$ est l'optimum (maximum ou minimum) de $f(x)$ par rapport à x. Afin de représenter concrètement cette formule, considérons un problème basique de contrôle optimal avec $f_1 = f$ et $f_2 =$ identité

$$OPT_{x,y}\{f(x, y)\} = OPT_x\{f(x, OPT_y\{y\})\} \tag{2.3}$$

où

- x représente l'état actuel du système en fonction du temps (cellule parente)
- y représente le contrôle appliqué au système (une fonction de transition d'état $\theta(x)$) (spécialisation de la cellule)
- $f(x, y)$ représente l'état futur du système (cellule fille)

$$OPT_x\{x_{t+1}\} = OPT_x\{f(x_t, OPT_x\{\theta(x_t)\})\} \tag{2.4}$$

La programmation dynamique nous permet de trouver la suite de contrôles à appliquer afin d'atteindre l'état souhaité en partant de l'état souhaité et remontant de manière récursive à l'état actuel. Ce qui implique une connaissance a priori de l'état souhaité.

Le problème de la recherche des optima implique que l'optimum (état souhaité) est inconnu a priori. C'est ici que MCOO intervient en faisant le parcours inverse, c'est-à-dire, en partant de l'état actuel (connu) et en se dirigeant vers l'état souhaité en tolérant et rectifiant les erreurs de parcours.

Pour MCOO, $\theta(x)$ représente la variation introduite par une génération. Soit :

$$\boxed{OPT_x\{fobj(x_n)\} = OPT_x\{fobj(x_{n-1}), OPT_x\{fobj(x_{n-1} \pm \theta(x))\}\}} \tag{2.5}$$

où

- $\theta(x)$ représente l'erreur aléatoire introduite ;
- $fobj$ est l'évaluateur d'adéquation (fonction objectif + fonction de contrainte)

MCOO ne garantit pas que cette variation intermédiaire est optimale à cette date intermédiaire. Cependant, MCOO doit garantir que l'erreur de cette variation intermédiaire par rapport à la variation optimale tend vers 0 quand le nombre de pas augmente.

Cette garantie est offerte à travers la méthode de spécialisation par la réduction de l'espace de variabilité.

2.1.4 Preuve de convergence de MCOO

Considérons le paramètre de la taille de l'espace de variabilité des points chercheurs *spreadFactor*(sf) et le paramètre de sa réduction *spreadFactorDecay* (sfd) :

$$sf_n = \begin{cases} a \text{ tel que } a \in \mathbb{R} \text{ si } n = 0 \\ sf_{n-1} \times sfd \text{ tel que } sfd \in [0,1) \text{ si } n > 0 \end{cases} \tag{2.6}$$

$$\boxed{\lim_{n \to \infty} sf_n = 0} \tag{2.7}$$

Ceci garantit que, pour chaque cellule, MCOO converge vers un point, car, lorsque le nombre d'itérations tend vers l'infini, la taille de l'espace de variabilité des points chercheurs (les informatrices) tend vers 0.

La convergence vers l'optimum lui-même dépend de la fonction objectif et de la méthode de spécialisation qui doit permettre une exploration complète de l'espace de recherche (dans toutes les dimensions). Il est à noter qu'à cause de la nature récursive que cette suite confère à l'algorithme, si la méthode de spécialisation ne permet pas de correction d'erreur, les erreurs s'amplifient et MCOO converge vers un point autre que l'optimum.

2.2 Méthode d'application

2.2.1 Optimisation en espace continu

Il est ici montré comment MCOO peut être appliquée à la recherche et l'optimisation sur un espace continu.

La cellule est un point de l'espace de recherche.

La spécialisation s'effectue en générant une cellule dans le voisinage. L'évaluateur d'adéquation est la fonction objectif avec/sans la fonction des contraintes.

MCOO, grâce à son paramètre de réduction de l'espace de variabilité (*spreadFactorDecay*), transforme progressivement l'exploration en exploitation intensive

(recherche sur des espaces infinitésimaux) à l'image d'un effet de zoom optique. Ensuite, grâce à son paramètre de tolérance à l'erreur (*growthTolerance*), elle permet de sortir des optima locaux.

La spécialisation est la seule opération présente dans MCOO. Elle doit donc permettre une couverture parfaite de l'espace de recherche (grande exploration) afin de ne pas ignorer de solutions possibles.

L'exploitation est une caractéristique inhérente à la méthode, à cause de la phase de spécialisation qui génère des cellules de plus en plus adaptées à la tâche (cellules dans un espace de recherche de plus en plus petit). Il reste donc, pour ajouter l'exploration, à contrebalancer cette forte exploitation.

Pour ce faire, la méthode de spécialisation utilise une suite divergente qui est décrite ci-dessous.

Une suite comme celle ci-dessous accroît la précision et peut être utile pour la recherche sur les espaces infinitésimaux :

$$U_n = \begin{cases} x \text{ si } n = 0 \text{ tel que } x \in \mathbb{R} \\ U_{n-1} \times sf_n \times rand(-1,1)) \text{ si } n \geq 1 \end{cases} \tag{2.8}$$

où

- U_n est une cellule spécialisée à l'itération n
- sf est la taille de l'espace de variabilité **spreadFactor**
- sfd est le coefficient de réduction de la taille de l'espace de variabilité **spreadFactorDecay**.La réduction de sf se fait dans la fonction **optimiser**
- *rand* est un générateur de nombres aléatoires

Trouvons la forme formelle de cette suite :

$U_0 = x$

$U_1 = U_0 \times sf_1 \times rand(-1,1) = x \times sf_0 \times sfd \times rand(-1,1)$

$U_2 = (x \times sf_0 \times sfd \times rand(-1,1)) \times sf_2 \times rand(-1,1)$

$= x \times (sf_0 \times sfd \times rand(-1,1)) \times (sf_1 \times sfd) \times rand(-1,1)$

$= x \times (sf_0 \times sfd \times rand(-1,1)) \times ((sf_0 \times sfd) \times sfd) \times rand(-1,1)$

$= x \times sf_0^2 \times sfd^2 \times rand(-1,1)^2$

...

$U_n = x \times sf_0^n \times sfd^n \times rand(-1,1)^n$

Puis, calculons sa limite quand n tend vers $+\infty$:

$$-1 < rand(-1,1) < 1 \implies -1 < rand(-1,1)^n < 1 \tag{2.9}$$

$$0 \leq sfd < 1 \implies 0 \leq sfd^n < 1 \tag{2.10}$$

$$-1 < sfd^n \times rand(-1,1)^n < 1 \tag{2.11}$$

$$|sfd^n \times rand(-1,1)^n| < 1 \implies \lim_{n\to\infty} (sfd^n \times rand(-1,1))^n = 0 \tag{2.12}$$

$$\lim_{n\to\infty} U_n = \lim_{n\to\infty} x \times \lim_{n\to\infty} sf_0^n \times \lim_{n\to\infty} (sfd^n \times rand(-1,1))^n \tag{2.13}$$

$$\lim_{n\to\infty} U_n \text{ n'est pas definie si } |sf_0| > 1 \tag{2.14}$$

$$\lim_{n\to\infty} U_n = 0 \text{ si } |sf_0| \leq 1 \tag{2.15}$$

Alternativement, une suite comme celle à (2.16) peut aussi être utilisée.

$$U_n = \begin{cases} x \text{ si } n = 0 \text{ tel que } x \in \mathbb{R} \\ U_{n-1} + sf_n \times rand(-1,1)) \text{ si } n \geq 1 \end{cases} \tag{2.16}$$

Ce qu'on note est que le taux de croissance $U_n - U_{n-1}$ dépend de la taille de l'espace de variabilité, *spreadFactor*(**sf**) et du facteur de réduction de la taille de l'espace de variabilité, *spreadFactorDecay*(**sfd**). Ce sont ces facteurs qui décident du pas de la progression de la recherche, et incidemment, de la vitesse de convergence de MCOO.

Plus l'âge de la cellule augmente, plus son **spreadFactor** diminue. Les cellules de plus en plus nouvelles se rapprochent donc très rapidement de leur cellule parente. Le pseudopode de la fonction de spécialisation dans le cas de l'optimisation en

espace continu est décrit en 9. Elle consiste à créer une cellule fille dans le voisinage de sa cellule parente.

Algorithm 9: MCOO - en espace continu

Input: *rand* un générateur aléatoire uniforme de nombres flottants

Input: *explorationFactor* nombre de nouvelles cellules à créer

```
1 function spécialiser(vieilleCellule, explorationFactor)
2     nouvellesCellules ← ∅;
3     k ← explorationFactor;
4     while k > 0 do
5         nouvelleCellule.coords ←
           vieilleCellule.coords + spreadFactor * rand(−1, 1);
6         nouvellesCellules ← nouvellesCellules.insrer(nouvelleCellule);
7         k ← k − 1;
8     return nouvellesCellules
```

2.2.2 Optimisation en espace discret

Nous voyons dans la suite comment MCOO peut être appliqué à la recherche et l'optimisation combinatoire.

La cellule est une combinaison de variables (aptitudes).

La spécialisation s'effectue en ajoutant une aptitude sélectionnée aléatoirement, à la cellule.

La recherche commence donc avec des cellules souches qui sont des solutions candidates constituées d'une seule variable initiale d'entrée (aptitude élémentaire), et chaque spécialisation y ajoute un élément, sélectionné aléatoirement. L'évaluateur d'adéquation est la fonction objectif avec/sans la fonction des contraintes.

Ce type problème doit par contre être relaxé afin de ne pas réessayer des possibili-

tés déjà évaluées (cf chapitre 4 de mise à l'épreuve). Rappelons qu'une relaxation est une technique qui permet de réduire l'espace de recherche. Nous suggérons la relaxation continue de Dantzig qui transpose les variables discrètes vers un domaine continu en créant une relation d'ordre entre elles. Elle consiste, concrètement à associer des coûts différents (parfois arbitraires) aux éléments du domaine (transformations ou aptitudes innées de l'organisme) et à les ordonner. Dantzig est celui à qui nous devons cette technique.

2.3 Limites

Le comportement de la procédure de recherche et d'optimisation repose entièrement sur les cellules souches de l'organisme (l'ensemble initial de variables) et le domaine des transformations qui sont représentées par les aptitudes innées de l'organisme. Cette limite inhérente à la méthode, peut entraîner un biais dans les résultats, qui est directement lié au choix des cellules souches. Un exemple de ce biais sera montré plus bas (cas de 0).

En outre, dû au caractère récursif de la procédure de recherche (cellule qui naît d'une certaine cellule, qui elle-même naît d'une autre cellule...), le choix de l'évaluateur d'adéquation (la fonction objectif) devient un paramètre qui intervient dans la direction de la progression de ladite procédure.

Une limite qui doit être prise en compte lors de l'application de la méthode proposée est le fait qu'elle cible des problèmes en horizon infini. Elle peut donc faire croître l'organisme indéfiniment, si sa taille n'est pas explicitement limitée. Puisque les ressources computationnelles disponibles sont finies, elles deviennent donc un paramètre à considérer, à l'image de GP, mais elle, contrairement à GP, supporte nativement les structures topologiques multi-chemin.

La propriété **âge** de la cellule permet à l'organisme de se débarrasser des cellules plus âgées. Cette suppression va à l'encontre du principe fondamental de la mé-

thode qui est de permettre la conservation de blocs réutilisables de solution. Dû à la limite des ressources computationnelles disponibles, cette variation peut être une alternative substantielle lors d'une recherche combinatoire.

2.4 Variantes de MCOO

2.4.1 La mort des cellules âgées

Cette variante consiste à purger l'organisme de cellules inutiles (dominées par toutes les autres cellules de l'organisme) et âgées. Elle est utilisée pour l'optimisation en espace continu dans le cadre de notre étude.

2.4.2 Les deux organismes

L'insertion ordonnée (en fonction du fitness) des cellules dans l'organisme est une opération très coûteuse ($O(n^2)$ pour le pire cas). Un moyen de limiter ce coût est de conserver l'ordre de dominance dans un organisme séparé constitué uniquement des meilleures cellules. Cet organisme doit évidemment être plus petit en taille que l'organisme global.

Le concept de MCOO a été présenté, et il a été montré comment il peut être appliqué à la recherche sur espace continu et à la recherche combinatoire. Il a été montré les limites inhérentes à sa nature, qui sont le choix des cellules souches et des aptitudes innées. Dans le chapitre suivant, nous montrons la forte convergence de MCOO par rapport à quelques méthodes heuristiques.

CHAPITRE III

VÉRIFICATION

3.1 Évaluation de la performance

Afin d'évaluer adéquatement la performance de la méthode proposée, nous la comparons à PSO et GSA face à des problèmes d'optimisation en espace continu. Concrètement, MCOO doit trouver l'optimum global des fonctions usuelles de benchmarking (Sphere, Rastrigin, Ackley...) dans un grand nombre de dimensions, ou s'en rapprocher plus que PSO et GSA. Cette expérimentation a pour but de vérifier que MCOO respectent les exigences de sa conception : s'extirper des optima locaux et se rapprocher plus vite de l'optimum global que GSA et PSO. Puis, nous faisons une étude comparative des temps d'exécution de MCOO versus PSO afin de montrer comment cette forte vitesse de convergence se traduit en temps d'exécution.

3.1.1 Méthodologie et protocole d'expérimentation

Dans un premier temps les résultats de MCOO sont comparés à ceux de deux principales méthodes heuristiques du moment : GSA et PSO par rapport à des fonctions de benchmarking tirées de deux articles (Rashedi *et al.*, 2009) et (Lim et Haron, 2013). Le premier est celui des auteurs de GSA et montre les résultats de GSA qui sont meilleurs à ceux de PSO en termes de vitesse de convergence

(la distance entre l'optimum trouvé et le véritable optimum global de la fonction de benchmarking par rapport au nombre d'itérations). Le deuxième article fait une simple étude comparative de PSO et d'autres techniques d'optimisations et montre comment PSO est meilleur en termes de vitesse de convergence. Ces deux articles sont utilisés, car en montrant que MCOO est meilleur à GSA en termes de vitesse de convergence, on montre que MCOO est meilleur que PSO et les techniques mentionnées dans l'étude de Lim et Haron, par transitivité. Dans un deuxième temps, nous mettons en exergue les problèmes rencontrés.
Rashedi et al montrent expérimentalement que GSA a une vitesse de convergence plus grande que celle de PSO (Rashedi *et al.*, 2009), autrement dit que GSA se rapproche plus près de l'optimum global que PSO, pour un même nombre d'itérations. Rappelons que leur expérience se déroule sur 1000 itérations.

MCOO est comparé aux résultats obtenus par ces chercheurs en utilisant les mêmes variables et la même métrique de référence qui est la vitesse de convergence.
Pour ce faire, les limites imposées sur les variables communes par Rashedi et al seront reportées pour notre expérience. Elles ont été choisies arbitrairement par Rashedi et al.

- le nombre de dimensions des variables est fixé à 30 ;
- la population est fixée à 50 ;
- le nombre d'itérations est fixé à 1000 ;
- le nombre d'essais est fixé à 30.

Les fonctions de benchmarking utilisées par Rashedi et al sont répertoriées dans la table 3.1 et la table 3.2. Ces fonctions ont été choisies par Rashedi et al., car elles montrent plusieurs difficultés que rencontrent les méthodes de recherches et d'optimisation.

Tableau 3.1: Fonctions unimodales

	Fonction	Limites	Optimum
f1	$\sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	$[0]^n$
f2	$\sum_{i=1}^{n} \|x_i\| + \prod_{i=1}^{n} \|x_i\|$	$[-10, 10]^n$	$[0]^n$
f3	$\sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]^n$	$[0]^n$
f4	$max\{\|x_i\|, 1 \leq i \leq n\}$	$[-100, 100]^n$	$[0]^n$
f5	$\sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30, 30]^n$	$[1]^n$
f6	$\sum_{i=1}^{n} ([x_i + 0.5])^2$	$[-100, 100]^n$	$[0]^n$
f7	$\sum_{i=1}^{n} ix_i^4 + random[0, 1)$	$[-1.8, 1.8]^n$	$[0]^n$

Tableau 3.2: Fonctions multimodales

	Fonction	Limites	Optimum
f8	$\sum_{i=1}^{n} -x_i sin(\sqrt{\|x_i\|})$	$[-500, 500]^n$	$[420.96]^n$
f9	$\sum_{i=1}^{n} [x_i^2] - 10cos(2\pi x_i) + 10$	$[-5.12, 5.12]^n$	$[0]^n$
f10	$-20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2})$ $-exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$	$[-32, 32]^n$	$[0]^n$
f11	$\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]^n$	$[0]^n$

Lim et Haron comparent les performances de PSO à celles de l'algorithme génétique traditionnel (GA) et de l'évolution différentielle (DE). Leur expérience inclut le temps d'exécution. Cette métrique ne peut être utilisée pour notre expérience, car les conditions exactes de l'expérimentation et les ressources computationnelles exactes utilisées par Lim et Haron n'ont pas été précisées. Les fonctions de benchmarking utilisées par Lim et Haron sont répertoriées dans la table 3.3 et la table 3.4.

Tableau 3.3: Fonctions de benchmarking utilisées par Lim et Haron - limites et optima

Fonction	Limites	Optimum
Sphere	$[-5.12, 5.12]^n$	$[0]^n$
Ackley	$[-30, 30]^n$	$[0]^n$
Rastrigin	$[-5.12, 5.12]^n$	$[0]^n$
Zakharov	$[-5, 10]^n$	$[0]^n$
Axis parallel hyper-ellipsoid	$[-5.12, 5.12]^n$	$[0]^n$
Griewank	$[-600, 600]^n$	$[0]^n$
Sum of Different Power	$[-1, 1]^n$	$[0]^n$
Schwefel Double Sum	$[-65.536, 65.536]^n$	$[0]^n$
Quartic with Noise	$[-1.28, 1.28]^n$	$[0]^n$
Michalewitz	$[0, \pi]^n$	$[-0.966n]^n$

Tableau 3.4: Fonctions de benchmarking utilisées par Lim et Haron - équations

Fonction	Équation
Sphere	$\sum_{i=1}^{n} x_i^2$
Ackley	$-a\exp(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)) + a + \exp(1)$
Rastrigin	$10n + \sum_{i=1}^{n}[x_i^2 - 10cos(2\pi x_i)]$
Zakharov	$\sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$
Axis Parallel Hyper-Ellipsoid	$\sum_{i=1}^{n} ix_i^2$
Griewank	$\sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$
Sum of Different Power	$\sum_{i=1}^{n} \lvert x_i\rvert^{i+1}$

Schwefel Double Sum	$\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j)^2$
Quartic with Noise	$\sum_{i=1}^{n}ix_i^4 + random[0,1)$
Michalewitz	$-\sum_{i=1}^{n}sin(x_i)sin^{2m}(\frac{ix_i^2}{\pi})$

Pour leur expérience, ils choisissent les valeurs suivantes :

- le nombre de dimensions des variables est fixé à 30
- la population est fixée à 40
- le nombre d'itérations est fixé à 2000
- le nombre d'essais est fixé à 10

Plus de détails dans la table 3.5.

Paramètres utilisés par Lim et Haron

Tableau 3.5: Paramètres utilisés par Lim et Haron

Parameter	Value
Number of generation	2000
Population size, n	40
Number of dimension, d	30
GA Crossover probability	0.7
GA Mutation probability	0.01
DE Crossover probability	0.8
DE Differential weight	0.3
PSO Inertia weight	0.7
PSO Acceleration constant	2
PSO Random number	[0,1]
PSO Maximum velocity	half of range
PSO Maximum position	range of dataset

Number of testing	10

Les variables propres à MCOO sont fixées comme suit :

- le pourcentage de spécialisation,**specialisationRate** est fixé à 0.2
- la tolérance à la mauvaise qualité (fitness) des solutions nouvelles, **growthTolerance** est fixé à 0.2
- le facteur d'exploration de l'espace de variabilité,**explorationFactor** est fixé à 10
- la taille initiale de l'espace de variabilité,**spreadFactor** est fixé à taille de l'espace de recherche divisé par 4
- le coefficient de réduction de l'espace de variabilité,**spreadFactorDecay** est fixé à 0.1

3.1.2 Résultats

Comparaison des résultats de MCOO avec les résultats de Rashedi et al.

Les résultats obtenus sont répertoriés dans la table 3.6 et la table 3.7. Rappelons que "unimodale" veut dire "avoir un seul mode", ce qui se traduit dans notre cas par un seul optimum. Conjointement, "multimodale" signifie la présence de plusieurs optima.
La solution moyenne est la moyenne de toutes les solutions sur les 30 essais. La solution médiane est la médiane de toutes les abscisses des solutions sur les 30 essais. La qualité moyenne est la moyenne des qualités de solutions sur les 30 essais.
Plus le résultat est près de l'optimum réel en annexe (souvent zéro et souvent un minimum), plus le résultat est bon. Les optima obtenus par MCOO sont plus près des optima globaux que ceux obtenus par GSA et PSO pour le même nombre

d'itérations (1000).

Tableau 3.6: Comparaison MCOO, PSO, GSA sur des fonctions unimodales utilisées par Rashidi et al.

fonction	statistiques	PSO	GSA	MCOO
f1	Solution moyenne	1.8×10^{-3}	7.3×10^{-11}	2.08×10^{-164}
	Solution médiane	1.2×10^{-3}	7.1×10^{-11}	1.01×10^{-166}
	Fitness moyenne	5.0×10^{-2}	2.1×10^{-10}	0
f2	Solution moyenne	2.0	4.03×10^{-5}	0
	Solution médiane	1.9×10^{-3}	4.07×10^{-5}	0
	Fitness moyenne	2.0	6.9×10^{-5}	0
f3	Solution moyenne	4.1×10^{3}	0.16×10^{3}	2.12×10^{-3}
	Solution médiane	2.2×10^{3}	0.15×10^{3}	5.05×10^{-167}
	Fitness moyenne	2.9×10^{3}	0.16×10^{3}	0
f4	Solution moyenne	8.1	3.7×10^{-6}	0
	Solution médiane	7.4	3.7×10^{-6}	0
	Fitness moyenne	23.6	8.5×10^{-6}	0
f5	Solution moyenne	3.6×10^{4}	25.16	0.18
	Solution médiane	1.7×10^{3}	25.18	1.1×10^{-2}
	Fitness moyenne	3.7×10^{4}	25.16	22.54
f6	Solution moyenne	1.0×10^{-3}	8.3×10^{-11}	-0.5
	Solution médiane	6.6×10^{-3}	7.7×10^{-11}	-0.5
	Fitness moyenne	0.02	2.6×10^{-10}	7.21×10^{-32}
f7	Solution moyenne	0.04	0.018	-3.13×10^{-4}
	Solution médiane	0.04	0.015	-1.72×10^{-23}
	Fitness moyenne	1.04	0.533	1.25×10^{-6}

Tableau 3.7: Comparaison MCOO, PSO, GSA sur des fonctions multimodales utilisées par Rashidi et al.

fonction	statistiques	PSO	GSA	MCOO
f8	Solution moyenne	-9.8×10^3	-2.8×10^3	202.74
	Solution médiane	-9.8×10^3	-2.6×10^3	420.92
	Fitness moyenne	-9.8×10^3	-1.1×10^3	-9.41×10^3
f9	Solution moyenne	55.1	15.32	-8.06×10^{-12}
	Solution médiane	56.6	15.42	-7.81×10^{-14}
	Fitness moyenne	72.8	15.32	0
f10	Solution moyenne	9.0×10^{-3}	6.9×10^{-6}	-2.29×10^{-19}
	Solution médiane	6.0×10^{-3}	6.9×10^{-6}	-3.2×10^{-22}
	Fitness moyenne	0.02	1.1×10^{-5}	4.44×10^{-16}
f11	Solution moyenne	0.01	0.29	3.55×10^{-10}
	Solution médiane	0.0081	0.04	4.44×10^{-20}
	Fitness moyenne	0.055	0.29	0

Les résultats de PSO et GSA sont extraits de l'article de Rashedi et al. tandis que les résultats de MCOO proviennent de notre expérimentation. Rappelons que nous utilisons une métrique commune qui est la vitesse de convergence.

NB : 0 comme résultat est un arrondi par l'environnement computationnel quand l'ordre de grandeur devient trop petit (Ex : 1×10^{-164})

Pour la fonction 8 en table 3.2, le coefficient d'exploration a été augmenté à 100 afin de permettre une meilleure couverture de l'espace de recherche.

En outre, nous émettons des réserves sur certains résultats obtenus pour la fonction 6, car l'optimum se trouve à $[-0.5]^n$ et MCOO le trouve sur les 30 essais, cependant GSA semble trouver une solution moyenne et médiane de l'ordre de 10^{-11} pour une fitness moyenne de 2.6×10^{-10}, une approximation de $[0]^n$.

Comparaison des résultats de MCOO avec les résultats de Lim et Haron

Les résultats, répertoriés dans la table 3.6, représentent l'optimum trouvé par les différentes fonctions, pour le même nombre d'itérations (1000). Plus le résultat est près de l'optimum réel en annexe (souvent zéro), meilleur il est.
Les optima obtenus par MCOO sont plus près des optima globaux que ceux obtenus par GSA et PSO pour le même nombre d'itérations.

Tableau 3.8: Comparaison MCOO, PSO, DE, GA sur des fonctions de benchmarking utilisées par Lim et Haron

Func.	Actual Optimum	Method	Min	Max	Average
Sphere	0	GA	0.0008	0.1978	0.0535
		DE	0.0040	0.3654	0.0932
		PSO	0.0099	0.0719	0.0237
		MCOO	0	0	0
Ackley	0	GA	0.0169	3.6098	0.6513
		DE	0.5545	4.5842	2.2593
		PSO	0.8398	2.3545	1.7527
		MCOO	4.44×10^{-16}	4.44×10^{-16}	4.44×10^{-16}
Rastrigin	0	GA	0.0563	19.1326	4.6809
		DE	5.0154	12.3525	9.7992
		PSO	46.1047	116.1930	79.3988
		MCOO	0	0	0
Zakharov	0	GA	0.2972	1.6587	0.7395
		DE	0.0649	1.9861	0.6328
		PSO	0.0613	0.6920	0.2835

		MCOO	0	0	0
Axis parallel hyper-ellipsoid	0	GA	0.0019	1.5947	0.4844
		DE	0.0032	0.7700	0.2245
		PSO	0.1685	1.1863	0.4755
		MCOO	0	0	0
Griewank	0	GA	2414.3400	2420.5900	2416.5120
		DE	91.0157	206.6490	164.3646
		PSO	361.0020	1081.0000	621.9848
		MCOO	0	0	0
Sum of Different Power	0	GA	1.0000E-06	5.4100E-04	1.0300E-04
		DE	8.9361E-11	1.9694E-06	3.1227E-07
		PSO	1.6500E-16	2.4500E-14	5.2400E-15
		MCOO	0	0	0
Schwefel Double Sum	0	GA	1.0000E-06	5.4100E-04	1.0300E-04
		DE	8.9361E-11	1.9694E-06	3.1227E-07
		PSO	1.6500E-16	2.4500E-14	5.2400E-15
		MCOO	0	0	0
Quartic with Noise	0	GA	0.0827	0.4930	0.2535
		DE	0.0054	0.1037	0.0242
		PSO	0.0342	0.0982	0.0578
		MCOO	1.19×10^{-7}	5.35×10^{-6}	1.84×10^{-6}
Michalewitz	-28.98	GA	-24.9448	-22.3162	-23.9686
		DE	-23.7439	-17.0515	-20.0908
		PSO	-22.9425	-14.9033	-19.0958
		MCOO	-21.73	-15.48	-19.31

3.1.3 Discussion

Comme l'illustrent les tableaux de résultats 3.7 et 3.8, MCOO, pour le même nombre d'itérations (1000), se rapproche plus de l'optimum global que GSA. Il dépasse donc GSA en termes de vitesse de convergence, et PSO, par la même occasion.

Il est à noter que les paramètres de MCOO utilisés pour l'expérience ne lui permettent pas d'échapper aux optima locaux de la fonction de Michalewitz cf table 3.8.

Des graphes de la convergence rapide de MCOO sont montrés en figure 3.1 et figure 3.2.

Le biais des transformations

Dans le cas de la fonction 4 table 3.1, La croissance de l'organisme requiert une spécialisation qui génère une cellule qui est plus proche de l'optimum $[0]^n$ que sa cellule parente, avec une condition très restrictive qui est que cette différence de proximité doit être notable sur toutes les dimensions simultanément. Une spécialisation qui utilise des transformations aléatoires sur chaque dimension séparément ne suffit plus afin de garantir la progression vers l'optimum en un temps fini. Une autre forme de spécialisation, qui applique la même transformation (multiplication par un facteur aléatoire) simultanément à toutes les dimensions de la cellule, a été donc ajoutée.

Ainsi, le choix de la spécialisation de la cellule a une influence directe sur le processus de recherche de l'optimum.

Le biais des cellules souches

Prenons l'exemple où le centre de l'espace de recherche est choisi comme cellule souche, et que l'optimum se retrouve en ce point, le meilleur résultat est obtenu à la première itération. Un autre exemple serait une spécialisation dont les trans-

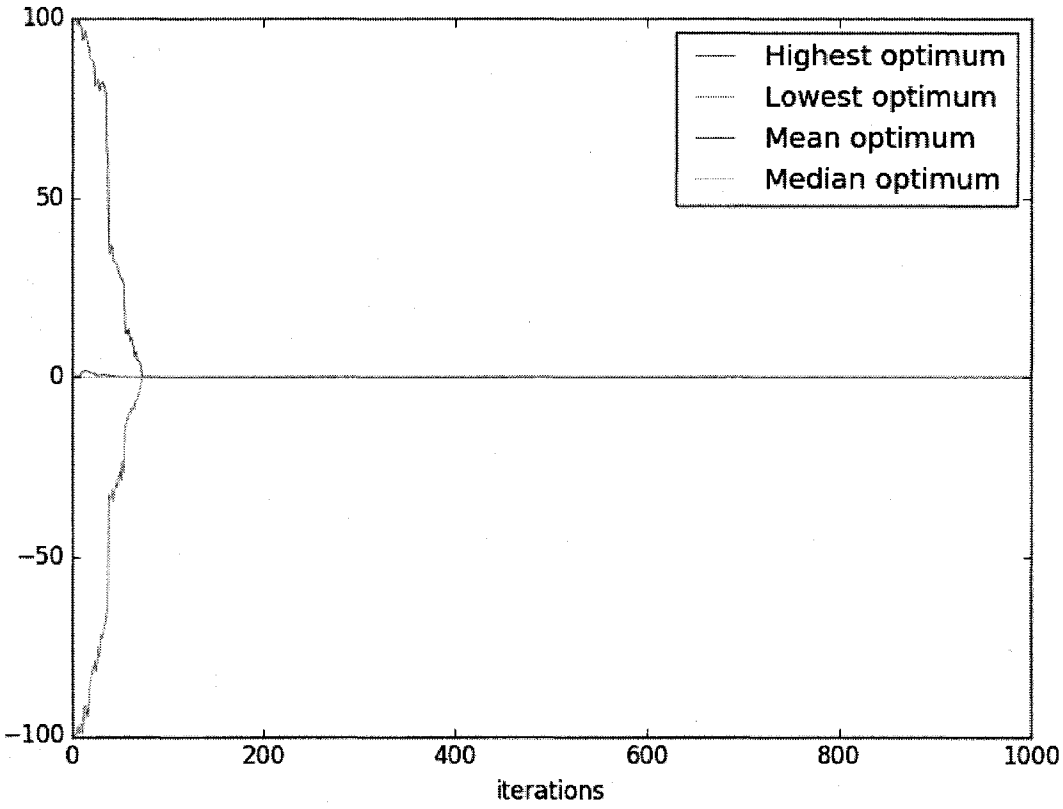


Figure 3.1: Convergence vers l'optimum pour la fonction 1 (-100 à 100, 1000 itérations)

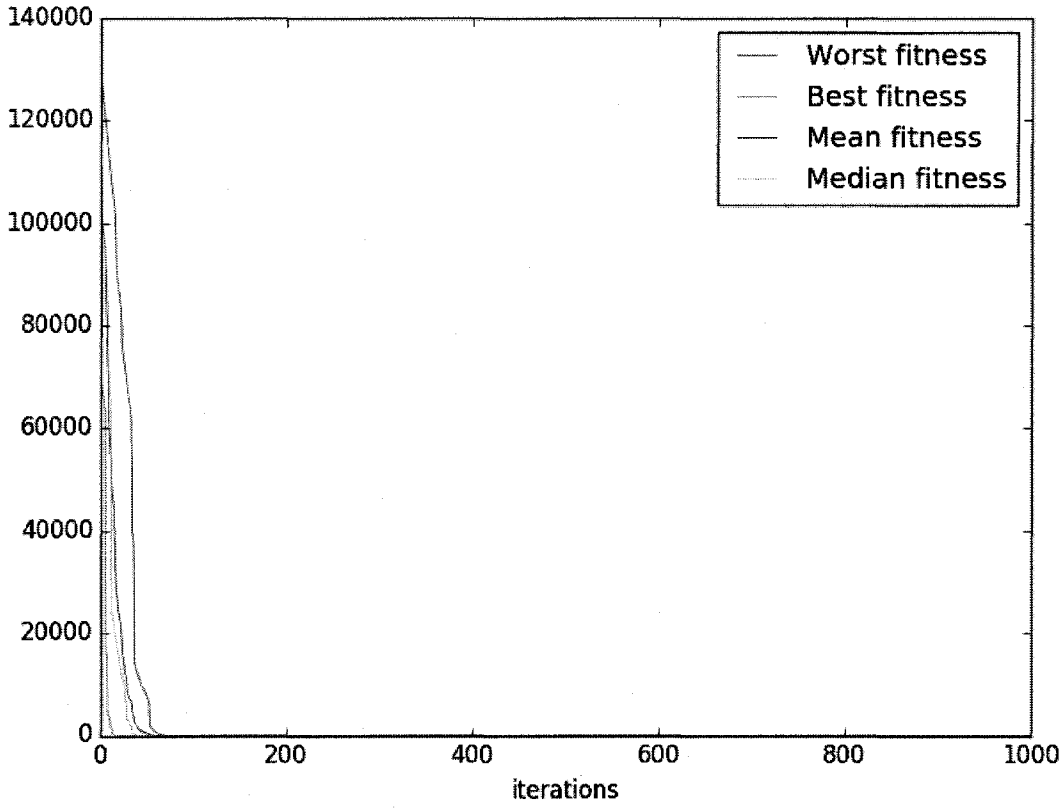


Figure 3.2: Convergence des fitness pour la fonction 1 (1000 itérations)

formations n'offrent que des résultats positifs, conjuguée avec une cellule souche à valeur négative ou un optimum négatif. Ces cas sont des exemples de biais introduits dans la progression vers l'optimum.
Bien que MCOO converge rapidement vers l'optimum, il est judicieux de montrer l'impact réel en termes de temps d'exécution.

3.2 Temps d'exécution

Après avoir montré la vitesse de convergence de MCOO, il nous incombe de montrer ce qu'elle représente en termes d'efficacité de la recherche. Pour ceci, nous utilisons les temps d'exécution comme métrique afin que le lecteur puisse constater le gain de l'utilisation de MCOO.
Puisque les temps d'exécutions des algorithmes qui utilisent des processus stochastiques ne sont pas une métrique parfaite de la vitesse de convergence, ils doivent être mesurés sur plusieurs essais, afin de s'assurer de leur relative constance, et ainsi être concluants.
Afin de comparer la performance de MCOO en termes de temps d'exécution, des essais sont conduits et comparés pour une recherche sur $[-1000, 1000]$ en (2) deux dimensions de la fonction objectif $y = x_1^4 - 2x_2x_1^2 + x_2^2 + x_1^2 - 2x1 + 5$.
Les essais se déroulent dans le même environnent Python sur une machine *Intel(R) Core(TM) i5 CPU M 560 @ 2.67GHz*. La librairie *pyswarm* indépendante, est utilisée pour obtenir les résultats de PSO.

MCOO met environ cinq fois moins de temps pour atteindre un résultat de même précision que PSO. Les résultats comparés en termes de temps d'exécution dans la table 3.9.
MCOO est en effet plus rapide que PSO, cependant, une mauvaise paramétrisation peut être néfaste à sa performance. Dans la partie suivante, nous étudions

Méthode	solution	qualité	Temps d'exécution(μs)
PSO	1.00001058, 0.99999144	4.000000001	**0.150**
PSO	1.00005143, 1.00010438	4.00000000265	0.150
PSO	0.99964552, 0.99922939	4.00000012947	0.146
PSO	0.99998725, 0.99997366	4.00000000016	0.169
PSO	0.99998921 0.99995567	4.00000000063	0.158
MCOO	1.00000397, 0.99999618	4.0000000001	**0.039**
MCOO	1.00000138, 1.00000295	4.0000000000	0.036
MCOO	0.99999866, 1.00000044	4.0000000000	0.038
MCOO	1.00000004, 0.99998793	4.0000000001	0.043
MCOO	0.99999063, 0.99998742	4.0000000001	0.037

Tableau 3.9: Comparaison des temps d'exécution de MCOO, PSO

l'influence des paramètres sur la performance de MCOO.

3.3 Comportement par rapport aux paramètres

Dans cette section, nous étudions l'influence des paramètres de MCOO sur son comportement. Le but de cette partie est d'essayer de prédire le comportement de MCOO, malgré le facteur aléatoire qui lui est inhérent.

3.3.1 Paramètres et éléments influents

Les différents paramètres de l'algorithme sont :

— L'âge maximal de l'organisme **maxAge**

— La taille maximale de l'organisme **maxSize**

— Le pourcentage de meilleures cellules à spécialiser **specializationRate**

— La tolérance à la mauvaise qualité (fitness) d'une nouvelle cellule par rap-

port à la meilleure cellule de l'organisme **growthTolerance**

- Le nombre de nouvelles cellules à créer **explorationFactor**
- La taille de l'espace de variabilité des nouvelles cellules **spreadFactor**
- Le coefficient de réduction de l'espace de variabilité des nouvelles cellules **spreadFactorDecay**
- La taille des données d'entrée
- Le générateur de nombre aléatoire utilisé

Les différentes variables des résultats sont :

- La qualité du résultat (mesurée par la fitness)
- Le temps d'exécution (mesurée en millisecondes)
- La taille finale de l'organisme

Les facteurs suivants, bien que susceptibles d'influencer les résultats obtenus par MCOO, demeurent spécifiques aux instances des problèmes.

- choix de la fonction objectif
- choix de l'ensemble des transformations

Dans ces figures, les points représentent des résultats d'exécution, les droites, des régressions linéaires sur l'ensemble des points et les unités de temps sont en ms.

3.3.2 Influence des paramètres avec une distribution uniforme

Âge maximal de l'organisme

L'âge maximal de l'organisme, qui correspond au nombre maximal d'itérations, a une influence sur la qualité (fitness) des résultats obtenus en permettant un raffinement progressif de la recherche.

Le pourcentage de cellules à spécialiser

Le pourcentage de cellules à spécialiser est le pourcentage de meilleures cellules de l'organisme qui peuvent être candidates à la spécialisation. Il est corrélé à la qualité (fitness) fig 3.3a. Le temps d'exécution croît avec le pourcentage de cellules à spécialiser, et la qualité des cellules décroît légèrement. fig 3.3b

Le facteur d'exploration

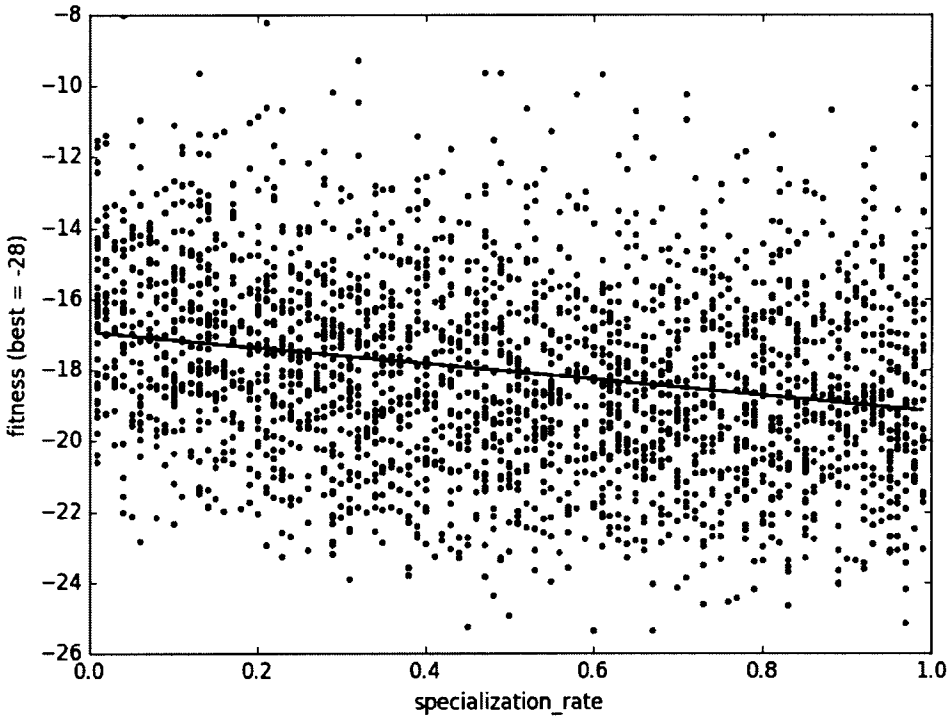
Le facteur d'exploration est le nombre de nouvelles cellules spécialisées créées pour chaque cellule à spécialiser. Le temps d'exécution croît avec le facteur d'exploration, et la qualité des cellules décroît. fig 3.4a, fig 3.4b. Son utilisation exige un compromis entre la qualité et le temps d'exécution.

La tolérance à la mauvaise qualité (fitness) de l'organisme

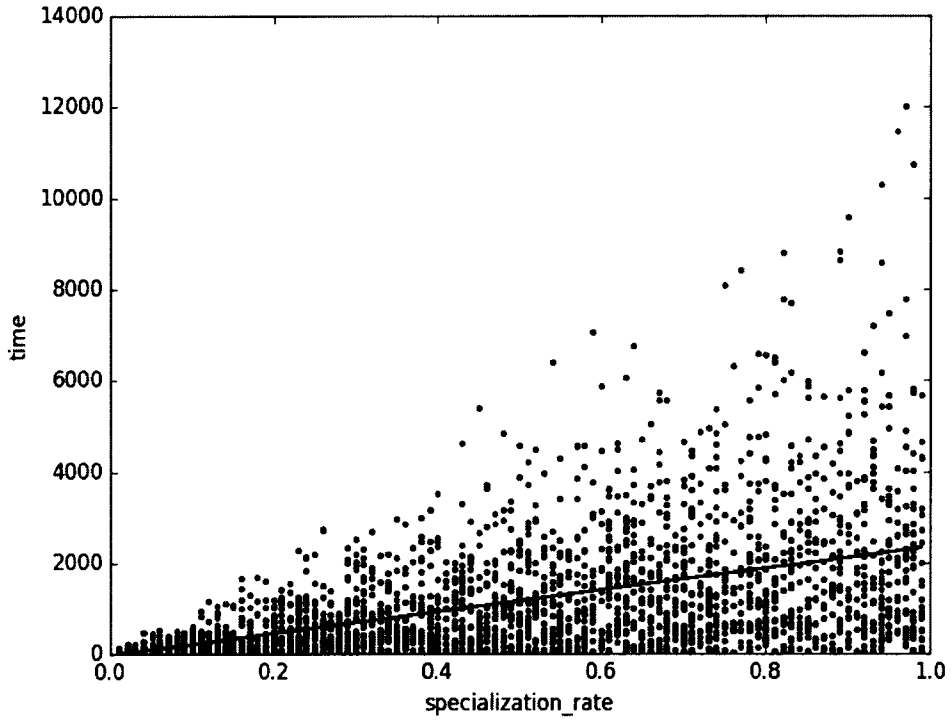
La tolérance à la mauvaise qualité (fitness) est le paramètre qui gouverne l'ajout de nouvelles cellules à l'organisme. Il permet, de par sa valeur, l'ajout d'une cellule dont la différence de qualité (fitness) avec la meilleure cellule actuelle est tolérable. Il a une légère incidence négative sur la qualité (fitness) des résultats fig 3.5a. Cependant, il a une influence moins grande sur le temps d'exécution fig 3.5b.

Les résultats obtenus par MCOO par rapport à GSA montrent que la proposition répond aux exigences qui ont donné lieu à sa conception, soit, échapper aux optima locaux, et converger vers l'optimum global plus rapidement que GSA et PSO. La technique de génération spontanée de cellule dans le voisinage lui confère une vitesse de convergence plus grande. Ceci est dû au fait que, plus l'optimum est proche, plus il y a de points chercheurs (informatrices) qui apparaissent dans son voisinage. Il est aussi montré, comment MCOO se comporte par rapport à ses paramètres. Ces résultats montrent que le concept théorique de MCOO

répond aux exigences de sa conception (recherche d'optima pour des fonctions de benchmarking comme Rastrigin). La question se pose de savoir s'il est capable de répondre efficacement à des problèmes plus concrets de recherche combinatoire.

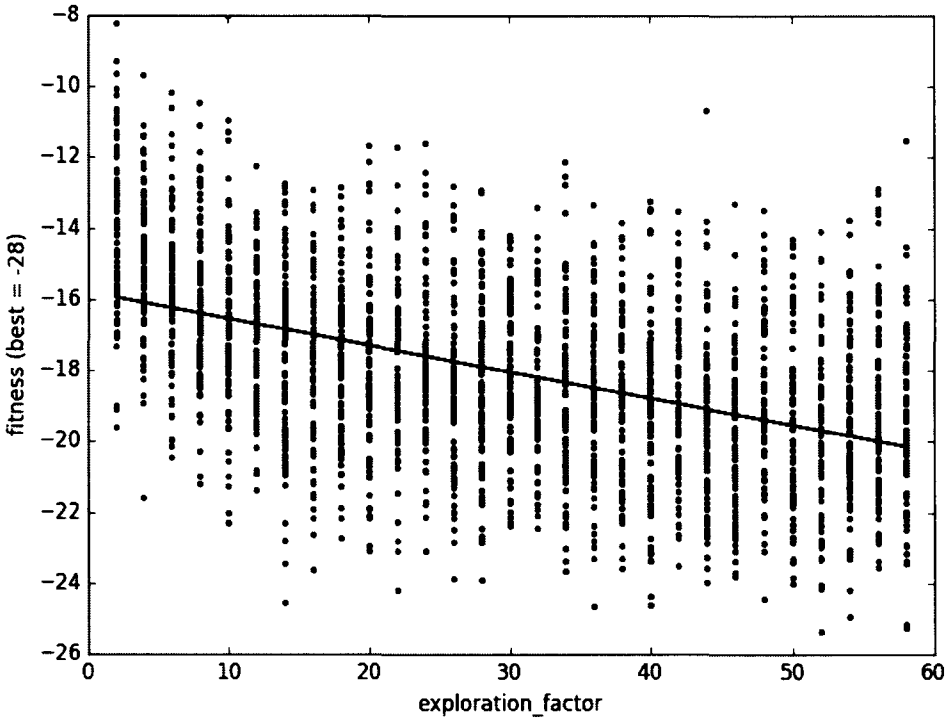


(a) sur la qualité (fitness)

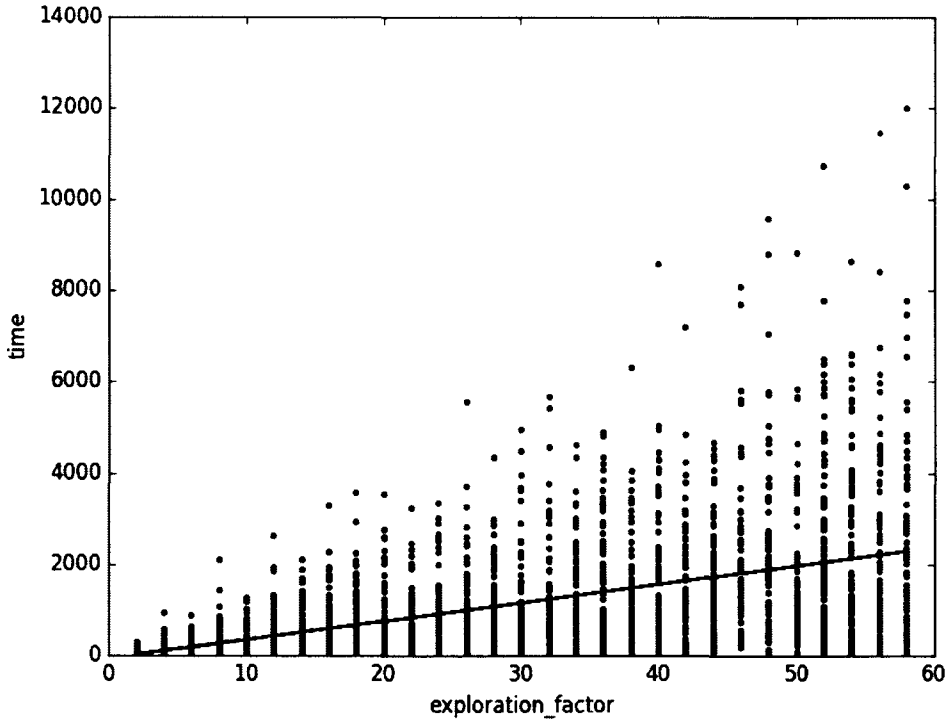


(b) sur le temps d'exécution

Figure 3.3: Influence du pourcentage de cellules à spécialiser

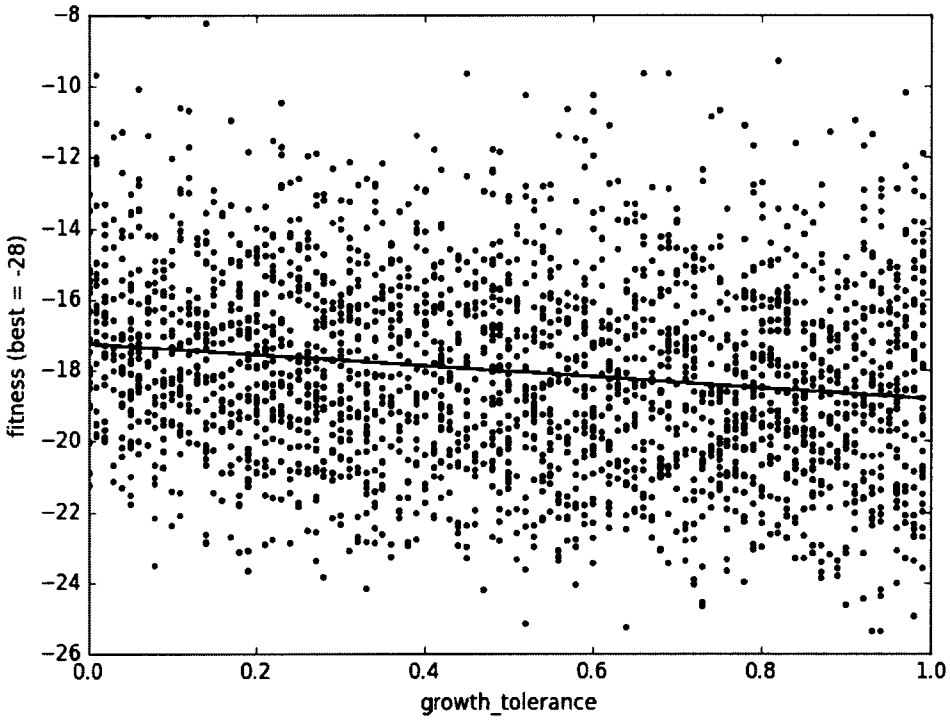


(a) sur la qualité (fitness)

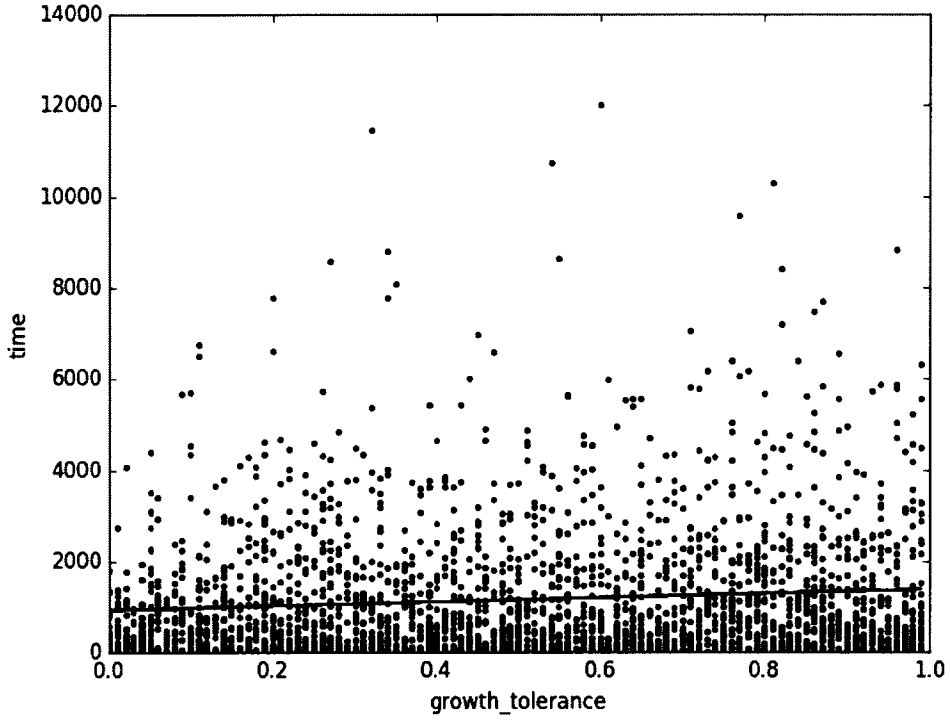


(b) sur le temps d'exécution

Figure 3.4: Influence du facteur d'exploration



(a) sur la qualité (fitness)



(b) sur le temps d'exécution

Figure 3.5: Influence de la tolérance à la mauvaise qualité (fitness)

CHAPITRE IV

VALIDATION

Dans cette section, nous mettons MCOO à l'épreuve avec un problème d'optimisation combinatoire avec quelques applications simples. Le but de cette partie est de montrer que le concept théorique de MCOO est nativement applicable à des problèmes de recherche combinatoire.
La méthodologie que nous utilisons consiste à valider que MCOO atteint effectivement la solution optimale ou satisfaisante (qui est connue d'avance) pour les problèmes choisis. Ensuite MCOO est appliquée à un problème de composition de transformations et un problème réel : l'ordonnancement des RSUs (défini plus loin).

4.1 Optimisation combinatoire

Nous choisissons trois problèmes NP-difficile, et vérifions que MCOO atteint les résultats attendus.

4.1.1 Choix du problème

Notre choix s'est arrêté sur le problème du sac à dos (knapsack problem), car c'est un problème prouvé comme étant NP-difficile par Martello et Toth (Martello et Toth, 1990).

Il est globalement défini comme suit : étant donné un sac de capacité C et une liste de N objets ayant chacun un poids W et un profit P. L'objectif est de trouver le sous-ensemble d'objets qui maximise le profit total, tout en ayant un poids total qui ne dépasse pas la capacité du sac.
Il existe plusieurs variantes du problème du sac à dos. Cependant, dans cette étude, nous utilisons une variante très restrictive du "Bounded Knapsack Problem (BKP)", le "0-1 knapsack", qui impose une limite d'au plus une copie par objet.

4.1.2 Représentation et méthodologie

L'application de MCOO au problème de sac à dos est faite comme suit :
Les objets ont chacun un profit $P(i)$ et un coût $W(i)$. La cellule est l'ensemble des objets choisis, soit une liste.
La fonction objectif de la cellule est définie comme suit :

$$fo = \begin{cases} 0, \forall i \sum_{i=0}^{n} W(i) > C \\ \sum_{i=0}^{n} P(i) \end{cases} \tag{4.1}$$

où n est le nombre de transformations identités de la cellule.
30 essais sont conduits sur trois instances du problème du sac à dos, ayant deux différents niveaux de difficulté. Le premier et le deuxième, plus faciles, ont des entrées de taille 10 et 8 respectivement. Le troisième, plus demandant sur les ressources computationnelles a une entrée de taille 24 et une capacité de 6404180. Plus de détails en appendice B.

4.1.3 Résultats et discussion

Pour les problèmes "01-Knapsack" non relaxés (les éléments du domaine ne sont pas ordonnés et le seuil n'est pas déterminé) suivants, le pourcentage de spécialisation **specializationRate** et la tolérance à la mauvaise qualité (fitness) **grow-**

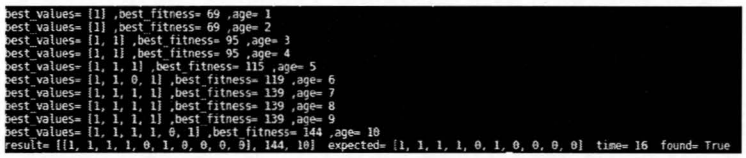


Figure 4.1: Exécution du problème 1 avec relaxation

thTolerance sont mis a 1.0 afin de conserver tous les résultats et de les faire intervenir dans la spécialisation.

Sans aucune relaxation, sur les meilleurs essais, l'optimum global est trouvé pour les problèmes 1 et 2 en un nombre d'itérations de l'ordre de 4×10^3 et de 6×10^5 pour le problème 3. Sur les pires essais, il n'est pas trouvé.

La relaxation continue de Dantzig [1] est utilisée afin de permettre une spécialisation qui augmente la performance de l'algorithme. La solution optimale est donc trouvée en un nombre d'itérations corrélé à la taille maximale de combinaison, soit 10 pour les problèmes 1 et 2 (taille 10) et 27 pour le problème 3 (taille 24), car la spécialisation utilise une forme de l'algorithme "branch-and-bound". Un exemple d'exécution est montré sur la figure 4.1. À chaque itération un élément est ajouté à chaque solution pour générer ses filles.

4.2 Optimisation de composition des transformations

Afin de tester le concept de MCOO appliqué à un problème de composition de transformations (plus de détails sur la définition de ce problème se trouve en Annexe A), le jeu de l'arithmétique a été utilisé. Il est défini comme suit : étant donné un ensemble fini de i nombres entiers n_i et quatre transformations ($addition = (a + b)$, $multiplication = (a * b)$, $soustraction = (a - b)$ et $division = (a/b)$) dé-

1. cf Martello et Toth, dans les documents de référence

finies sur cet ensemble, quelle composition c permet de trouver l'entier recherché n_r ?

Ce problème est choisi, car il est une version simplifiée du problème de la synthèse automatique des circuits. Il suffit de remplacer les opérateurs arithmétiques par des portes logiques et les entiers par des listes de booléens pour concevoir un circuit logique.

4.2.1 Représentation du problème

Afin de transposer ce problème en une représentation computationnelle, l'unité structurale de l'organisme de MCOO, est la cellule définie comme étant un vecteur de transformations (f1, f2...) ayant des entrées (en vert) et une sortie (en rouge) et un vecteur de données (d1, d2...), comme le montre la figure 4.2.

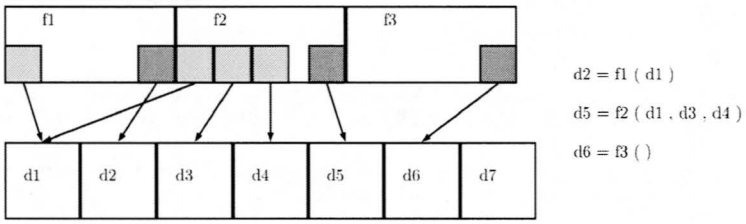


Figure 4.2: Structure d'une cellule de transformations

4.2.2 Résultats et discussion

Afin d'utiliser la relaxation de Dantzig, le domaine des transformations est ordonné selon $P(i)/W(i)$ où P représente le profit d'utilisation de la transformation et W le coût d'utilisation de la transformation. $P(i) = 1 \forall i$ arbitrairement, car les profits des transformations dépendent de leur sorties et ne peuvent donc être connus a priori.

Voici la liste des profits/coûts par opération, déterminés de façon arbitraire :

— addition : 0.05

— multiplication : 0.034

— soustraction : 0.02

— division : 0.032

La table 4.1 montre quatre traces d'exécution avec en entrée 1,2,7 et une valeur recherchée de 25. Dans les traces d'exécution, $x@y$ veut dire que la valeur x se trouve dans la case de données y. La figure 4.3 représente la structure topologique multi-chemin générée par MCOO pour la première trace d'exécution.

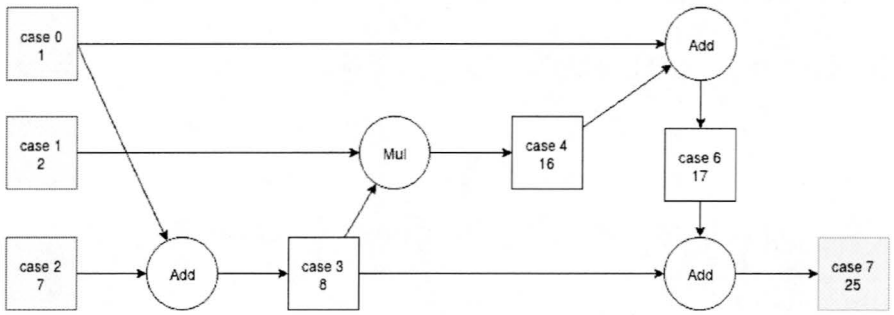


Figure 4.3: Structure topologique multi-chemin créée par la première trace d'exécution

Comme le montrent ces résultats, la solution optimale est atteinte. Il est à remarquer que la minimisation du nombre de transformations utilisées dans la solution n'a pas été ajoutée comme contrainte à la recherche.

MCOO a été présenté et expérimentalement évalué, par rapport à l'optimisation continue, combinatoire et au problème de la composition des transformations, dans la troisième section de cette partie, comme un outil de recherche et d'optimisation efficace en termes de convergence.

Les problèmes réels (de la vie courante) sont bien plus complexes qu'une simple suite d'opérations arithmétiques. La nouvelle question, qui se pose, est de connaître la performance de MCOO face à un problème réel : l'ordonnancement des Road Side Units.

Tableau 4.1: Composition de transformations trouvées

Composition de transformations	Temps d'exécution
8@3=add(1@0,7@2), 16@4=mul(2@1,8@3), 9@5=add(1@0,8@3), 17@6=add(16@4,1@0), 25@7=add(17@6,8@3)	12μs
9@3=add(2@1,7@2), 16@4=add(7@2,9@3), 23@5=add(7@2,16@4), 8@6=div(16@4,2@1), 11@7=add(9@3,2@1), 25@8=add(16@4,9@3)	295μs
14@3=add(7@2,7@2), 16@4=add(14@3,2@1), 9@5=add(2@1,7@2), 25@6=add(9@5,16@4)	57μs
8@3=add(1@0,7@2), 8@4=mul(1@0,8@3), 16@5=mul(2@1,8@4), 9@6=add(2@1,7@2), 18@7=mul(2@1,9@6), 25@8=add(18@7,7@2)	24μs

4.3 Application de MCOO à un problème réel : l'ordonnancement des RSUs

Les RSUs (*Road Side Units*) sont des bornes de communications sans-fil qui permettent des échanges de données avec des véhicules sur une route. Ils peuvent être reliés à une source d'énergie stable comme une centrale électrique et/ou être alimentés par des énergies renouvelables comme celle solaire. Cette étude tente de répondre à la question de savoir comment MCOO peut être appliquée au problème d'ordonnancement des RSUs. Dans la première partie, le problème de l'ordonnancement est présenté et formalisé. Dans la seconde, les simplifications apportées au problème sont présentées. Ces simplifications régissent notre expérimentation.

La représentation computationnelle de ce problème pour l'utilisation de MCOO est présenté dans la troisième partie. Les résultats sont présentés dans la dernière partie.

4.4 Présentation

Le problème de l'ordonnancement des RSUs consiste à allouer *optimalement* des créneaux de temps d'antenne sur des RSUs à des véhicules en mouvement (cf. (Atoui *et al.*, 2016)).
Le terme *optimalement* peut prendre plusieurs significations, cependant, puisque notre étude tente de répondre à la question de desservir le plus grand nombre de véhicules avec le moins d'énergie possible, *optimalement* est donc défini comme étant : **desservir le plus grand nombre de véhicules avec le moins d'énergie possible**. Rappelons que l'ordonnanceur est une entité distincte du RSU.

4.4.1 Formalisation

Étant donné un ensemble fini $M = \{r_1, r_2, r_3, ..., r_m\}$ de m RSUs, ayant chacune, une queue Q_m de créneaux de temps de taille maximale k_m, et un ensemble $N = \{v_1, v_2, v_3, ..., v_n\}$ de n véhicules, sur le tronçon de route couvert par l'ordonnanceur, disponibles pour l'allocation, quel ensemble d'allocations (r_m, v_n) permet de desservir le plus possible de véhicules avec le moins possible d'énergie ?

Objectif

l'objectif ainsi défini est donc constitué de deux critères :

- maximiser le nombre de véhicules desservis
- minimiser la quantité d'énergie utilisée

Soit $c_m(n, t)$ la quantité d'énergie nécessaire à un RSU m pour transmettre à un

véhicule n à la date t

Dans le cas où le RSU est incapable de choisir automatiquement la date de début de transmission, l'ordonnanceur doit se charger de cette tâche, et donc associer des dates de début et de fin de transmission à chaque allocation RSU-Véhicule.

La question devient :

Quelle **combinaison** a de l'ensemble de toutes les allocations possibles $Z = \{(r_1, v_1, t_{start}(1,1), t_{end}(1,1)), (r_1, v_2, t_{start}(1,1), t_{end}(1,1)), ...,$
$(r_1, v_n, t_{start}(1,n), t_{end}(1,n)), (r_2, v_1, t_{start}(2,1), t_{end}(2,1)), ...,$
$(r_m, v_n, t_{start}(m,n), t_{end}(m,n))\}$ permet de desservir le plus possible de véhicules avec le moins possible d'énergie ?

Dans le cas où le RSU est capable de choisir automatiquement la date de début de transmission, il existe deux modes d'opération à considérer :

— le RSU est responsable d'ordonner les données.

— le RSU n'est pas responsable d'ordonner les données.

Dans le premier cas, l'ordonnanceur est libéré de cette tâche et associe simplement les véhicules aux RSUs en fonction de la vitesse et de la position du véhicule au moment de la requête.

La question devient :

Quelle **combinaison** a de l'ensemble de toutes les allocations possibles $Z = \{(r_1, v_1), (r_1, v_2), ..., (r_1, v_n), (r_2, v_1), ..., (r_m, v_n)\}$ permet de desservir le plus possible de véhicules avec le moins possible d'énergie ?

Dans le deuxième cas, l'ordonnanceur doit aussi ordonner les données des véhicules en fonction de leur vitesse et de leur position au moment de la requête. La question devient :

Quel **arrangement** a de quel sous-ensemble $X \subseteq Z$ de l'ensemble de toutes les allocations possibles $Z = \{(r_1, v_1), (r_1, v_2), ..., (r_1, v_n), (r_2, v_1), ..., (r_m, v_n)\}$

Dans les deux cas, l'objectif premier de cet arrangement ou combinaison ultime est de maximiser la cardinalité $|V'|$ de l'ensemble des véhicules V' présents dans

a.

$$max(|V'|) \tag{OI}$$

et, minimise l'énergie totale utilisée C_m

$$min(C_m) \text{ tel que } C_m = \sum_{l \in V'} \int_{t_{start}(m,l)}^{t_{end}(m,l)} c_m(l,t)\,\mathrm{d}t \tag{OII}$$

où

$t_{start}(m,l)$ est la date de début de transmission du RSU m pour le véhicule l

$t_{end}(m,l)$ est la date de fin de transmission du RSU m pour le véhicule l

avec comme contrainte

$$\forall l \in V_i', \int_{t_{start}(m,l)}^{t_{end}(m,l)} c_m(l,t)\,\mathrm{d}t < e_m(t_{start}(m,l)) + \int_{t_{start}(m,l)}^{t_{end}(m,l)} h_m(t)\,\mathrm{d}t \tag{CI}$$

où

V_i' est l'ensemble des véhicules associés au RSU m

$h_m(t)$ est la quantité d'énergie récoltée par le RSU m à la date t

$e_m(t)$ est la quantité d'énergie disponible pour le RSU m à la date t

La cardinalité de l'ensemble O de tous les ordonnancements possibles est :

$$|O| = \sum_{l=1}^{|Z|} A_{|Z|}^{l} \tag{4.2}$$

Justification de OII :

Bien que l'énergie soit renouvelable, elle n'est pas illimitée. Le modèle d'énergie renouvelable utilisé est une quantité d'énergie récoltée de manière aléatoire, en fonction du temps, variant entre 0 et une valeur maximale.

L'hypothèse est qu'à la date t, la batterie a un certain niveau d'énergie, et à la date $t+1$, elle se rechargera d'une quantité aléatoire (donc imprévisible) inférieure à sa capacité de récolte maximale. À la même date $t+1$, cette récolte (gain) est contrebalancée par la consommation (perte) qui dépend des allocations et donc du temps.

L'ordonnanceur n'a aucun contrôle sur les gains, mais il a un contrôle absolu sur les pertes, donc, la solution réelle à ce problème revient à trouver l'ordonnancement (ensemble d'allocations) qui minimise les pertes, c'est-à-dire, l'énergie consommée. D'où le caractère incontournable de l'économie d'énergie dans ce problème d'ordonnancement de RSU.

4.4.2 La nature dynamique du problème

Le fait que les véhicules sont en mouvement à des vitesses variables, la date de début transmission (et donc de fin de transmission) pour chaque véhicule, varie pour chaque allocation et par la même occasion, la quantité d'énergie requise pour la transmission à chaque véhicule, pour chaque allocation.
L'objectif OII décrit la propriété écoénergétique du système. L'utilité de cette propriété écoénergétique prend tout son sens, lorsque l'ordonnanceur opère en temps réel (quand le nombre futur de véhicules est inconnu), car le plus d'énergie les RSUs ont en réserve, le plus de véhicules ils pourront desservir dans le futur. Cette propriété devient donc une condition incontournable à une grande disponibilité du système.

4.4.3 Relaxation

Compte tenu de la nature dynamique du problème, un véhicule, entre dans le rayon de portée d'un RSU entre des dates $t_{entering\_range}(m,n)$ et sors de sa portée $t_{leaving\_range}(m,n)$ tel que $t_{entering\_range}(m,n) \leq t_{start}(m,n)$ et $t_{leaving\_range}(m,n) > t_{end}(m,n)$. Il est donc inutile de considérer toutes les dates comme choix potentiels pour les allocations de Z.

4.5 Application

Cette partie de l'étude, dans un premier temps, démontre comment MCOO peut être appliqué au problème d'ordonnancement des RSUs et dans un deuxième temps, évalue ses performances. La méthodologie d'évaluation de performance choisie est une comparaison des résultats de MCOO avec un solutionneur de programmation linéaire.

4.5.1 Simplification

Utilisons comme référence un solutionneur de programmation linéaire, car, bien que le problème soit NP-difficile, il peut être résolu avec un solutionneur linéaire s'il est de petite taille (d'où le choix du nombre de RSUs, de véhicules, etc.). Supposons que le RSU m est capable de sélectionner automatiquement les données à transmettre au véhicule n. l'objectif d'ordonnancement des données n'est donc plus une contrainte. Le problème devient un simple ordonnancement temporel des RSUs et non des blocs de données par RSU.

Supposons ainsi que l'ordonnanceur est responsable de la sélection les dates de transmission.

Pour des raisons de simplicité, supposons que les transmissions de tous les RSUs répondent à une même loi d'atténuation et possèdent toutes la même durée de créneau de temps $t_c = 1$ seconde.

Aussi, nous supposons que la date de fin de transmission t_{end} est toujours égale à la date de début $t_{start} + t_c$.

En discrétisant le temps, l'objectif (OII) devient :

$$min(C_m) \text{ tel que } C_m = \sum_{l \in V'} c_m(m, l, t) \qquad \text{(OII)}$$

Utilisons le modèle d'atténuation de Rappaport (Rappaport, 2001)

$$c_m(m,n,t) = P_{tx}(m,t) * TS_i = \frac{P_{rx}(j,t) * TS_i}{z_{0,m} * [\frac{d_{0,m}}{d(m,n,t)}]^{\gamma}} = \frac{N_0 * (2^{\frac{D}{B}} - 1) * TS_i}{z_{0,m} * [\frac{d_{0,m}}{d(m,n,t)}]^{\gamma}} \quad (4.3)$$

où

TS_i est la durée du créneau de temps pour le RSU m

$P_{tx}(m,t)$ est la puissance de transmission du RSU m à la date t

$P_{rx}(j,t)$ est la puissance de réception du véhicule n à la date t

$d_{0,m}$ est la distance de référence pour le RSU m

$z_{0,m}$ est l'atténuation de la propagation à $d_{0,m}$ pour le RSU m

$d(m,n,t)$ est la distance entre la RSU m et le véhicule n à la date t

D est la quantité de données (en bits) transmises

B est la bande passante du canal

γ est l'exposant d'atténuation de la propagation

N_0 est la puissance de bruit entre le RSU m et le véhicule n

4.5.2 Les contraintes

Suivent les contraintes additionnelles que nous choisissons :

- un véhicule doit être desservi au moins une fois (CII).
- Un RSU ne peut desservir qu'un seul véhicule à la fois : c'est-à-dire qu'un créneau de temps c dans n'importe quel RSU, ne peut être alloué qu'à un seul véhicule (CIII).

$$|V'| \geq |V| \quad \text{(CII)}$$

$$\forall i, \forall (v_m, v_n) \in V_i', [t_{start}(m,v_m), t_{end}(m,v_m)] \cap [t_{start}(m,v_n), t_{end}(m,v_n)] = \emptyset \quad \text{(CIII)}$$

4.5.3 Forme canonique du problème simplifié

Le problème peut être exprimé en problème de programmation linéaire comme suit :

Maximiser :

$$\sum_{m\in M,n\in N,t\in T}(m,n,t) \tag{OI}$$

Minimiser :

$$\sum_{m\in M,n\in N,t\in T} c_m(m,n,t) \tag{OII}$$

Sujet à :

$$\sum_{m\in M,t\in T}(m,n,t) \leq 1,\ \forall n \in N \tag{CII}$$

$$\sum_{n\in N}(m,n,t) \leq 1,\ \forall m \in M, t \in T \tag{CIII}$$

$$\sum_{m\in M,n\in N,t\in T} c_m(m,n,t) < e_m(t-1) + h_m(t),\ \forall t \in T \tag{CI}$$

4.6 Représentation computationnelle pour MCOO

Afin de pouvoir résoudre ce problème grâce à MCOO, il nous faut le transposer dans une représentation optimale et compréhensible par MCOO, donc definir.

4.6.1 La cellule

La cellule qui est l'unité structurale de MCOO est constituée d'un vecteur de listes de triplets (représentant les allocations : RSU, véhicule, créneau de temps).

4.6.2 La spécialisation

La méthode de spécialisation choisie consiste en trois étapes :

— ajouter une allocation (a) aléatoire à un RSU sélectionné de manière aléatoire
— modifier chaque allocation en utilisant **spreadFactor**
— ajouter l'allocation (a) déjà créée dans une cellule vide (sans allocation)

4.6.3 L'adéquation à la tâche (fitness)

La qualité des cellules (fitness) est un couple de deux critères équipotents qui sont dérivés des objectifs : le nombre de véhicules servis et l'énergie totale utilisée. Ses éléments sont comparés selon la dominance de Pareto.

4.7 Analyse expérimentale

4.7.1 Méthodologie

30 essais sont conduits avec les variables de la table 4.2. Les variables de la table 4.3 sont utilisées pour MCOO.

4.7.2 Résultats

Avec une durée d'étude de 2000 créneaux (qui correspond à la longueur de la route divisée par la durée d'un créneau de temps), le système dessert aisément 50 véhicules (100%). Cependant, avec une durée d'étude de 20 créneaux (afin de simuler le stress), le maximum atteint par MCOO est 43 (idem pour PuLP) avec moyenne de 41, voir fig 4.6.

4.7.3 Discussion

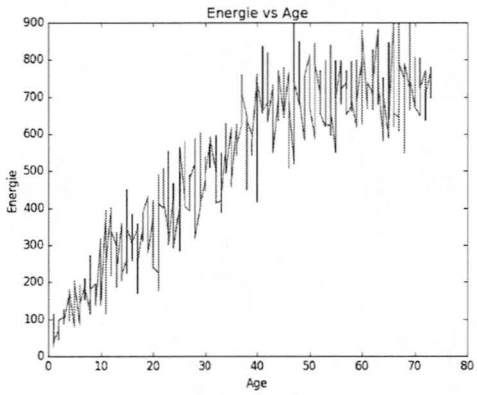
Le nombre de véhicules desservis est linéairement relié à l'âge des cellules. Ceci est dû à la méthode de spécialisation choisie qui fait croître la cellule d'une assi-

Variables	Valeur	Unité
Durée d'un créneau de temps	1	s
Longueur de la route	2000	m
Rayon de couverture d'une RSU	400	m
Nombre de RSUs	3	
Énergie maximale récoltée (/temps)	3	J
Niveau initial d'énergie	100	J
Vitesse moyenne	25	m/s
Exigence de données des véhicules (D)	1000000	bits
Bande passante du canal (B)	10000000	Hz
Exposant d'atténuation (γ)	3	
Distance de référence (d_0)	1	m
Atténuation à d_0 (z_0)	1	
Puissance de bruit dans le récepteur (N_0)	0.001	W

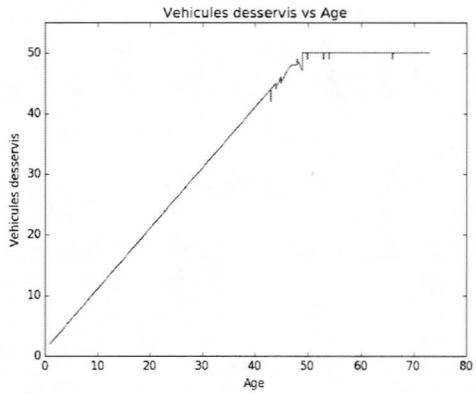
Tableau 4.2: Variables du problème

Paramètres	Valeur
Age maximal de l'organisme	100
Taille maximale de l'organisme	20
Pourcentage de spécialisation	1
Tolérance a la mauvaise qualité	1
Facteur d'exploration	30
Coefficient de réduction du facteur d'exploration	0.5

Tableau 4.3: Variables utilisées pour MCOO

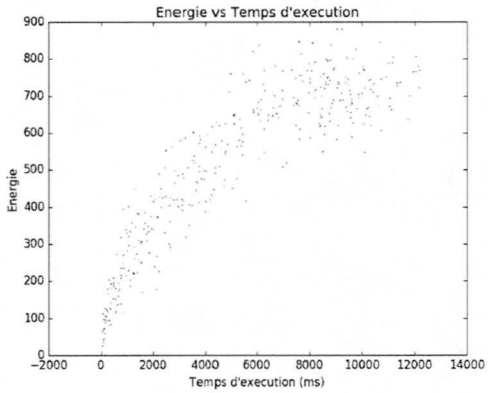


(a) sur l'énergie

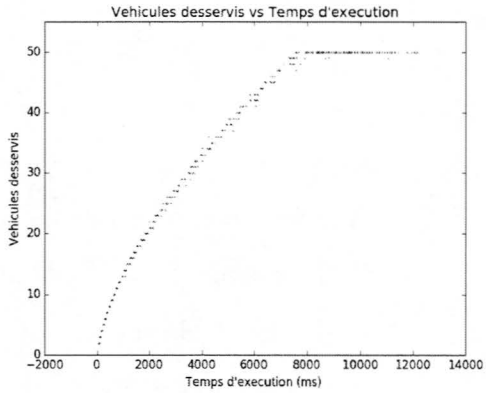


(b) sur le nombre de véhicules desservis

Figure 4.4: Influence de l'âge des cellules



(a) sur l'énergie



(b) sur le nombre de véhicules desservis

Figure 4.5: Influence du temps d'exécution

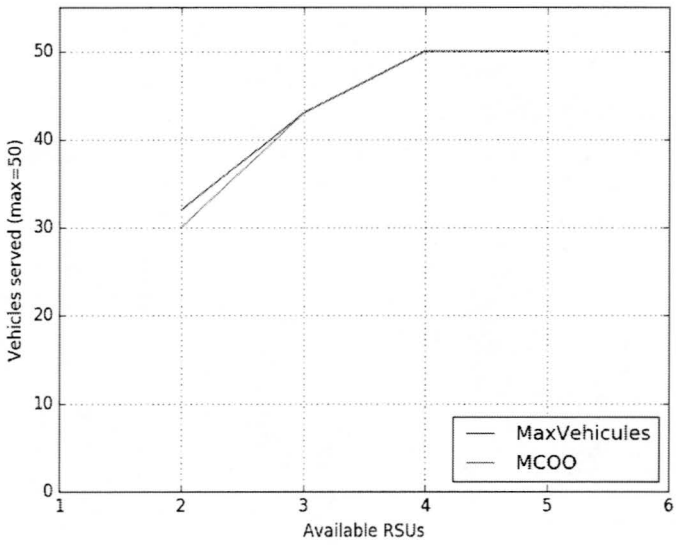


Figure 4.6: Comparaison de MCOO avec les résultats du solutionneur de programmation linéaire.

gnation.

Le temps d'exécution est exponentiellement relié au nombre de véhicules desservis. Le temps d'exécution croît exponentiellement avec l'âge. Ce qui est attendu, car plus le nombre de véhicules augmente (avec l'âge), plus les possibilités de spécialisation deviennent nombreuses, et plus le temps utilisé pour l'évaluation des cellules augmente.

La nature instable de la courbe de l'énergie est due au fait qu'il est évalué en second et dépend donc du nombre de véhicules desservis.

Il a été montré comment MCOO peut être appliqué au problème d'ordonnancement des RSUs en définissant la cellule comme un ensemble d'allocations RSU-Vehicule-Timeslot. Les performances en termes de convergence vers l'optimum de MCOO se rapprochent de celles de la méthode de programmation linéaire. Ce qui montre que MCOO est outil valide pour des applications du monde réel.

CONCLUSION

Nous avons proposé une approche heuristique inspirée de la croissance des êtres multicellulaires que nous nommons Optimisation par Organisme Multicellulaire ou MCOO (Multi-cellular Organism Optimization).
Les résultats obtenus dans le chapitre 3 montrent que MCOO une vitesse de convergence plus grande que GSA et PSO. Ceci se traduit en des temps d'exécution plus courts, comme cinq fois moins que PSO, pour un même degré de précision.
En outre, les résultats de son application au problème des RSUs au chapitre 4, se rapprochant de ceux du solutionneur de programmation entière, et dépassant ceux de PSO (qui plafonne à 80%), en termes de nombres de véhicules desservis pour 3 RSUs et l'atteignant pour 4 RSUs, montrent que non seulement MCOO est applicable aux problèmes réels, mais il est aussi un outil performant de recherche et optimisation.
Cependant, il a été aussi montré au chapitre 3, qu'une mauvaise paramétrisation peut nuire à sa vitesse, comme le biais introduit par le choix des cellules souches et des aptitudes innées.

MCOO a été présenté et expérimentalement prouvé comme un outil de recherche et d'optimisation efficace en termes de convergence et nous avons aussi mis en exergue ses limites.
Parallèlement à l'application de MCOO à l'ordonnancement des RSUs (Road Side Units), MCOO a été aussi appliqué au modèle proposé, afin de concevoir automatiquement des circuits logiques (étude qui n'est pas rapportée dans ce mémoire).
Compte tenu de sa grande flexibilité, les applications de MCOO sont multiples,

ainsi, dans des travaux futurs, nous utiliserons MCOO pour la découverte de motifs ("patterns") comme des configurations neuronales.

APPENDICE A

LE PROBLÈME DE LA COMPOSITION DES TRANSFORMATIONS

Une transformation est une opération t définie sur un domaine D qui prend $x \in D$ et lui associe une image $y \in t(D)$ avec $t(D) = \{t(x) \mid x \in D\}$ d'après Eric Weisstein (Weisstein, 1999).

Des exemples de transformations sont la dilatation, la réflexion, la rotation, le *mapping* (correspondance)....Elles peuvent opérer sur des espaces fonctionnels, des espaces topologiques...

Une composition de transformations est une transformation obtenue par l'imbrication (*nesting*) de deux ou plusieurs transformations, Plus formellement, soit E un domaine, une composition ou loi de composition d'arité n sur E est une transformation de E^n vers E avec $n \in \mathbb{N}$.

Il est entendu, dans la suite, par niveau de composition, le nombre de niveaux d'imbrication de la composition.

Exemple :$niveau(f(x)) = 0$, $niveau(f(g(h(x)))) = 2$

Il est entendu par problème de la composition des transformations, le problème de **la synthèse automatique et de l'optimisation des compositions de transformations**. Le terme "problème de la composition des transformations" est utilisé dans la suite du document pour des raisons de brièveté.

Un problème d'optimisation est un problème de recherche de la meilleure solution possible.

Un problème d'optimisation sur espace continu est un problème d'optimisation dont l'espace des variables est un continuum.

Le problème de la composition de transformations est un cas particulier de problème d'optimisation d'un espace dynamique de variables discrètes vers un espace de solutions discret, continu ou semi-continu [1] Exemple : $(]-\pi, -2[\cup -1, 0\cup]1, \infty])\cap \mathbb{R}$, car l'évaluation de la solution candidate repose souvent sur l'évaluation de l'extrant de la composition finale qui peut appartenir à un espace discret, continu ou discontinu.

Ce problème est celui ciblé par MCOO. Ce type de problème est caractérisé par un espace de variable avec une dimensionnalité dynamique (toujours croissante), car une solution candidate devient une nouvelle variable.

Le problème peut être défini comme suit :

Soit P l'ensemble de toutes les propriétés possibles. Une propriété, qui est tout ce qui caractérise un objet mathématique ou physique, est un concept abstrait représentant une qualification ou quantification (Ex : nombres complexes, fonctions, propriétés physiques, espaces, structure topologique, qualités, états, ...)

Étant donné un sous-ensemble de i propriétés $S = \{s_1, s_2, s_3, ..., s_i\}$ tel que $S \subseteq P$, un ensemble fini de j transformations $T = \{t_1, t_2, t_3, ..., t_j\}$ tel que

$t_1 : U_{11} \times U_{12} \times U_{13} \times .... \times U_{1k_1} \rightarrow V_1,$

$t_2 : U_{21} \times U_{22} \times U_{23} \times .... \times U_{2k_2} \rightarrow V_2,$

...,

$t_j : U_{j1} \times U_{j2} \times U_{j3} \times .... \times U_{jk_j} \rightarrow V_j$

avec

$U_{11} \subseteq P, U_{12} \subseteq P, ..., U_{1k_1} \subseteq P, ..., U_{21} \subseteq P, ..., U_{jk_j} \subseteq P$

---

1. semi-continu : continu sur certaines parties de l'espace universel et discret sur une ou plusieurs autres

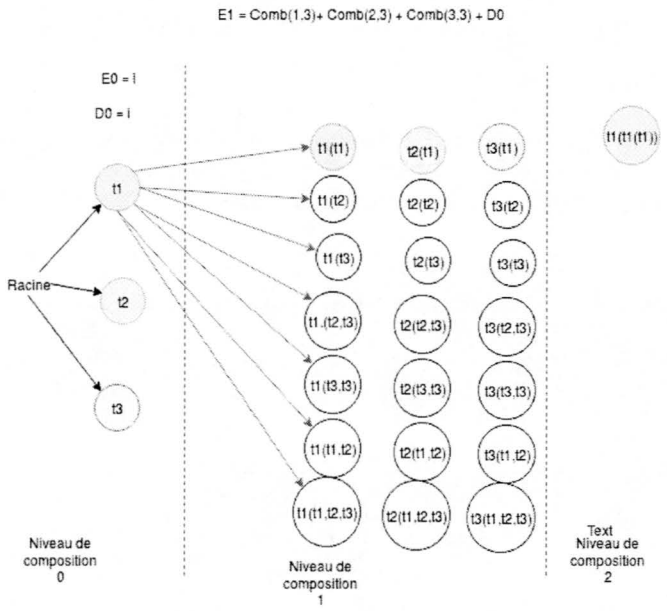


Figure A.1: Exemple de compositions possibles au degré un avec trois transformations à trois paramètres

et

$V_1 \subseteq P, V_2 \subseteq P, V_3 \subseteq P, ..., V_j \subseteq P,$

r une propriété telle que $r \in P$,

Quelle composition C (ou composition de compositions) des éléments de S transformés par les éléments de T, donne exactement la propriété r (satisfait), ou du moins, donne une propriété la plus similaire possible à r (minimise ou maximise) ?

Exemple : Prenons le cas de la conception automatique de circuits. Définissons un composant électrique comme étant un bloc fonctionnel agissant sur des quantités électriques en fonction du temps. Considérons la forme généralisée du bloc fonctionnel comme étant une transformation de quantités électriques.

Un circuit, étant un assemblage de composants électriques, est une structure topo-

logique de blocs fonctionnels, donc une structure topologique de transformations de quantités électriques.

La problématique fondamentale de l'assemblage est la structure topologique multi-chemin de ces transformations de quantités électriques.

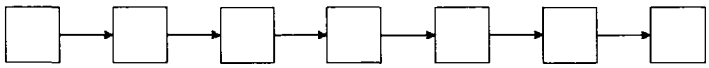


Figure A.2: Solution uni-chemin

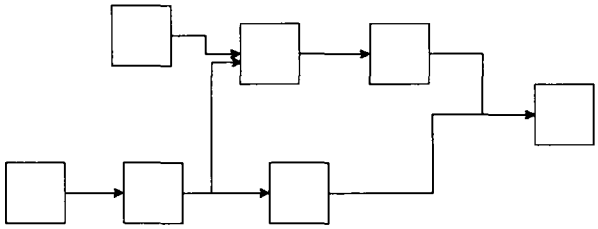


Figure A.3: Solution multi-chemin

APPENDICE B

EXEMPLE D'EXÉCUTION DE MCOO SUR DES PROBLÈMES DE SAC À DOS

Problème 1 : Succès

Capacité = 165

Profits = [92, 57, 49, 68, 60, 43, 67, 84, 87, 72]

Coûts = [23, 31, 29, 44, 53, 38, 63, 85, 89, 82]

Résultat attendu (référence) = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

Trouvé par MCOO = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

Problème 2 : Succès

Capacité = 104

Profits = [350, 400, 450, 20, 70, 8, 5, 5]

Coûts = [25, 35, 45, 5, 25, 3, 2, 2]

Résultat attendu (référence) = [1, 0, 1, 1, 1, 0, 1, 1]

Trouvé par MCOO = [1, 0, 1, 1, 1, 0, 1, 1]

Problème 3 : Succès

Capacité = 6404180

Profits = [825594, 1677009, 1676628, 1523970, 943972, 97426, 69666, 1296457, 1679693, 1902996, 1844992, 1049289, 1252836, 1319836, 953277, 2067538, 675367, 853655, 1826027, 65731, 901489, 577243, 466257, 369261]

Coûts = [382745, 799601, 909247, 729069, 467902, 44328, 34610, 698150, 823460, 903959, 853665, 551830, 610856, 670702, 488960, 951111, 323046, 446298, 931161, 31385, 496951, 264724, 224916, 169684]

Résultat attendu (référence) = [1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]

Trouvé par MCOO = [1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]

RÉFÉRENCES

Al-Saedi, W., Lachowicz, S., Habibi, D. et Bass, O. (2017). Pso algorithm for an optimal power controller in a microgrid. *IOP Conference Series : Earth and Environmental Science*, *73*(1), 012028.

Aote, S. S., Raghuwanshi, M. M. et Malik, L. (2013). A brief review on particle swarm optimization : Limitations & future directions. *International Journal of Computer Science Engineering*, *2*, 398–403.

Atoui, W. S., Salahuddin, M. A., Ajib, W. et Boukadoum, M. (2016). Scheduling energy harvesting roadside units in vehicular ad hoc networks.

Azadeh, A., Pashapour, S. et Zadeh, S. A. (2016). Designing a cellular manufacturing system considering decision style, skill and job security by NSGA-II and response surface methodology. *International Journal of Production Research*, *54*(22), 6825–6847.

Bailey, R. N., Garner, K. M. et Hobbs, M. F. (1997). Using simulated annealing and genetic algorithms to solve staff-scheduling problems. *Asia - Pacific Journal of Operational Research*, *14*(2), 27–43.

Banzhaf, W., Francone, F. D., Keller, R. E. et Nordin, P. (1998). *Genetic Programming : An Introduction : on the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

Bellman, R. (1957). Dynamic programming. *Princeton University Press.*

Das, S. et Suganthan, P. N. (2011). Differential evolution : A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, *15*(1), 4–31.

Deb, K., Pratap, A., Agarwal, S. et Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197.

del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Hernandez, J. C. et Harley, R. G. (2008). Particle swarm optimization : Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation*, *12*(2), 171–195.

Dorigo, M., Maniezzo, V. et Colorni, A. (1996). The ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26 (1) (1996), pp. 29–41.*

Farmer, J., Packard, N. et Perelson, A. (1986). The immune system, adaptation and machine learning. *Physica D, 2 , pp. 187–204.*

Heudin, J.-C. (2008). *Les créatures artificielles : Des automates aux mondes virtuels.* Odile Jacob.

Iwamoto, S. (1985). Sequential minimaximization under dynamic programming structure. *Journal of Mathematical Analysis and Applications*, *108*(1), 267 – 282.

Kennedy, J. et Eberhart, R. (1995a). Particle swarm optimization. *Proceedings of*

IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948.

Kennedy, J. et Eberhart, R. (1995b). Particle swarm optimization. Dans *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, 1942–1948 vol.4.

Kirkpatrick, S., Gelatto, C. et Vecchi, M. (1983). Optimization by simulated annealing. *Science, 220, pp. 671–680.*

KOZA, J. R., KEANE, M. A., STREETER, M. J., ADAMS, T. P. et JONES, L. W. (2004). Invention and creativity in automated design by means of genetic programming. *AI EDAM, 18*, 245–269.

Lim, S. P. et Haron, H. (2013). Performance comparison of genetic algorithm, differential evolution and particle swarm optimization towards benchmark functions. Dans *2013 IEEE Conference on Open Systems (ICOS)*, 41–46.

Lutton, J.-L. et Bonomi, E. (1986). Simulated annealing algorithm for the minimum weighted perfect euclidean matching problem. *Revue française d'automatique, d'informatique et de rechercheopérationnelle. Recherche opérationnelle*, *20*(3), 177–197.

Martello, S. et Toth, P. (1990). *Knapsack Problems : Algorithms and Computer Implementations.* New York, NY, USA : John Wiley & Sons, Inc.

Meyer-Nieberg, S. et Beyer, H.-G. (2012). *The Dynamical Systems Approach — Progress Measures and Convergence Properties*, Dans G. Rozenberg, T. Bäck, et J. N. Kok (dir.). *Handbook of Natural Computing*, (p. 741–814). Springer Berlin Heidelberg : Berlin, Heidelberg.

Mitten, L. G. (1964). Composition principles for synthesis of optimal multistage processes. *Operations Research*, *12*(4), 610–619.

Panda, S., Mohanty, B. et Hota, P. (2013). Hybrid bfoa–pso algorithm for automatic generation control of linear and nonlinear interconnected power systems. *Applied Soft Computing*, *13*(12), 4718 – 4730.

Patel, S. et Thakker, R. A. (2016). Automatic circuit design and optimization using modified pso algorithm. *Journal of Engineering Science and Technology Review*, *9*(1), 89–94.

Rappaport, T. (2001). *Wireless Communications : Principles and Practice* (2nd éd.). Upper Saddle River, NJ, USA : Prentice Hall PTR.

Rashedi, E., Nezamabadi-pour, H. et Saryazdi, S. (2009). GSA : A Gravitational Search Algorithm . *Information Sciences*, *179*(13), 2232 – 2248. Special Section on High Order Fuzzy Sets.

Ren, Y., Yu, D., Zhang, C., Tian, G., Meng, L. et Zhou, X. (2017). An improved gravitational search algorithm for profit-oriented partial disassembly line balancing problem. *International Journal of Production Research*, *55*(24), 7302–7316.

Russell, S. et Norvig, P. (1995). *Artificial Intelligence a Modern Approach*. Prentice Hall.

Weisstein, E. W. (1999). *CRC concise encyclopedia of mathematics*. Boca Raton, Flor : CRC Press.

Wilson, G. et Banzhaf, W. (2008). A comparison of cartesian genetic program-

ming and linear genetic programming. Dans *Proceedings of the 11th European Conference on Genetic Programming*, EuroGP'08, 182–193., Berlin, Heidelberg. Springer-Verlag.