09/2013 – M/INF

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
UNIVERSITE DES **S**CIENCES ET DE LA **T**ECHNOLOGIE **H**OUARI **B**OUMEDIENE (U.S.T.H.B)
Faculté d'Electronique et d'Informatique

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

En INFORMATIQUE

Spécialité : Systèmes Intelligents et Ingénierie du logiciel

*par : Mr **LASSOUAOUI Mourad***

Sujet

Approches Hyper-heuristiques pour le problème de la détermination du gagnant WDP

Soutenu publiquement le 23/06/2013 devant le jury :

Mme DRIAS Habiba	***Professeur à l'USTHB***	***Présidente***
Mlle BOUGHACI Dalila	***Maître de conférence/A à l'USTHB***	***Directrice de Mémoire***
Mr KECHID Samir	***Maître de conférence/A à l'USTHB***	***Examinateur***
Mr BOULIF Menouar	***Maître de conférence/A à l'UMBB***	***Examinateur***

REMERCIEMENTS

En premier lieu Je remercie ALLAH le tout puissant de m'avoir permis de mener à bien ce travail.

Je tiens à exprimer mes plus vifs remerciements et ma profonde gratitude à ma directrice de recherche ***Mlle Dalila Boughaci*** *pour m'avoir soumis un sujet si intéressant. Je la remercie également pour son orientation et ses conseils tout au long de mon parcours. Je lui suis entièrement reconnaissant d'avoir guidé mes premiers pas dans le monde de la recherche, et de m'avoir fait bénéficier de son expérience qui m'a été d'un très grand apport, aussi bien sur le plan scientifique que sur le plan humain.*

Je remercie vivement ***Mme Drias Habiba*** *pour l'honneur qu'elle m'a fait en acceptant de présider le jury de ma soutenance.*

Mes remerciements vont aussi à ***Mr Boulif Menouar*** *et* ***Mr Kechid Samir*** *d'avoir accepté de faire partie du jury.*

DÉDICACES

Je tiens tout d'abord à dédier ce travail à mes très chers parents, à qui je dois tout et sans qui rien de tout cela ne serait fait.

Sans oublier mes deux sœurs,

A ma grand-mère,

A toute ma famille(en particulier nounou),

A mes très chers amis pour tous leurs soutient, et surtout d'être mes amis (ils sauront se reconnaître).

J'exprime mes sentiments les plus profonds et leur dédie ce modeste travail.

Mourad.

RÉSUMÉ

Une hyper-heuristique est une approche d'optimisation utilisant un mécanisme de sélection dynamique des techniques d'optimisation qui servent à résoudre un problème d'optimisation. Une plateforme hyper-heuristique peut être vue comme une boîte noire constituée de deux couches : une couche basse et une couche supérieure. La couche basse est l'ensemble de techniques (méthodes heuristiques et/ou méta- heuristiques) du processus d'optimisation. La couche supérieure est l'hyper- heuristique qui sélectionne la technique d'optimisation la plus appropriée à appliquer en fonction d'informations non spécifique au problème d'optimisation à traiter.

Dans ce mémoire de magistère, nous nous intéressons au problème de la détermination du gagnant dans les enchères combinatoires appelé en anglais Winner Determination Problem (WDP). Le problème WDP dans les enchères combinatoires est un problème d'optimisation difficile dont l'objectif est de trouver un ensemble d'offres de clients participant à l'enchère combinatoire qui maximise le gain du vendeur.

Le but de notre travail consiste alors à proposer une approche hyper-heuristique pour la résolution efficace du problème WDP. Trois hyper-heuristiques ont été proposées et étudiées pour le WDP. Dans la première hyper-heuristique, le mécanisme de sélection d'une technique de couche basse à appliquer est basé sur une fonction de choix dite « *choice function* ». Cette dernière est calculée en fonction du temps d'exécution et la qualité de la solution trouvée par la technique d'optimisation. La deuxième hyper-heuristique se base sur une stratégie de choix aléatoire (*random choice*) pour la sélection d'une technique d'optimisation pour la recherche d'une solution à un niveau donné. Dans la troisième hyper-heuristique que nous appelons hyper heuristique stochastique, nous proposons une nouvelle stratégie de sélection. Le choix d'une technique d'optimisation est tributaire d'une probabilité P. Avec une probabilité P, le choix d'une technique d'optimisation se fait selon la stratégie de

***choice function*. Avec une probabilité 1-P, la technique d'optimisation est choisie selon la stratégie aléatoire « random choice ». La valeur de P est fixée empiriquement.**

Plusieurs expérimentations numériques ont été réalisées sur des benchmarks WDP de diverses tailles dans le but de tester et de prouver l'efficacité de nos approches. Les résultats trouvés sont très encourageants et montent l'intérêt de notre contribution. Les résultats trouvés par l'hyper-heuristique stochastique sont nettement meilleurs que ceux fournis par les autres hyper- heuristiques ainsi que ceux fournis par la méta- heuristique de recherche locale stochastique ou « *SLS* ».

Mots clés: Hyper-heuristique, Problème Du Gagnant dans les enchères combinatoires, recherche locale stochastique, enchères combinatoires.

SOMMAIRE

LISTE DES FIGURES

LISTE DES TABLE

LISTE DES SIGLES ET ABREVIATIONS

BoB	Branch-on-Bids
BoI	Branch-on-Items
CABoB	Combinatorial Auction Branch on Bid
CABRO	Combinatorial Auction BRanch and bound Optimizer
CASS	Combinatorial Auction Structured Search
CAMUS	Combinatorial Auction Multi-Unit Search
CF-HH	Choice Function Hyper-Heuristique
FAP	Frequency Assignment Problem
FCC	Federal Communication Commission
PDG	Problème de Détermination du Gagnant
SCF-HH	Stochastic Choice Function Hyper-Heuristique
SLS	Stochastic Local Search
R-HH	Random Hyper-Heuristique
WDP	Winner Determination Problem

INTRODUCTION GÉNÉRALE

La théorie des enchères est venue pour régler certains problèmes liés au domaine de l'intelligence artificielle, la recherche opérationnelle et l'économie. C'est l'un des sujets les plus largement étudiés en économie ces cinquante dernières années **[18].**

L'enchère la plus connue est l'enchère classique (ou ascendante) qui a vu sa réputation croître grâce à la vente de certains objets à des prix astronomiques. Mais l'enchère qui nous intéresse le plus est l'enchère combinatoire où dans ce type de pratique plusieurs objets sont mis en vente et les acheteurs soumettent une offre sur un ensemble d'objets qui les intéressent et qu'ils jugent complémentaires.

Dans ce type d'enchère, quand nous parlons d'objet nous ne faisons pas forcément allusion à un élément physique seulement, cela peut être aussi un service, une bande de fréquences radio, des ressources dans un système multi-agents,… En effet, plusieurs applications ont vu le jour afin de résoudre des problèmes réels dans l'industrie. Nous prenons comme exemple : l'attribution de licence radio dans les réseaux cellulaires, l'allocation des créneaux de décollage et d'atterrissage dans les aéroports, … **[18]** Ce qui veut tout simplement dire : quand nous avons un ensemble d'"*objets*" et un ensemble de personnes intéressées par l'acquisition de ceux-ci avec une soumission d'offre jugée complémentaire, alors ce problème peut être considéré comme un problème d'enchère combinatoire.

La détermination du gagnant dans une enchère combinatoire est un problème d'optimisation difficile, où le vendeur est amené à choisir plusieurs offres d'acheteurs en évitant au même moment de dépasser la quantité de chaque objet attribué aux acheteurs, provoquant ainsi des conflits entre eux.

Ce type de problème a souvent été traité à l'aide de méthode d'optimisation exacte (Branch&Bound,…) et de méthodes approchées (Recherche Taboue, Recherche Locale Stochastique, …). Dans notre travail, nous nous sommes intéressés au développement de méthodes de haut niveau d'abstraction faisant interagir plusieurs méthodes de recherche - heuristiques et/ou métaheuristiques- liées au problème. Ces méthodes ont pour nom *hyper-heuristiques*. L'idée principale d'une hyper-heuristique

est de compenser les faiblesses de certaines méthodes de recherche par la force des autres **[49]**. Le principe général des hyper-heuristiques est de sélectionner/générer une méthode heuristique à partir d'un ensemble de méthodes/composants du problème à traiter.

Le but de notre travail consiste à proposer une approche hyper- heuristique pour la résolution efficace du problème WDP. Trois hyper- heuristiques ont été proposées et étudiées pour le WDP. Dans la première hyper- heuristique, le mécanisme de sélection d'une technique de couche basse à appliquer est basé sur une fonction de choix dite « *choice function* ». Cette dernière se base dans ses calculs sur le temps d'exécution et la qualité de la solution trouvée de la méthode de recherche. La deuxième hyper-heuristique se base sur une stratégie de choix aléatoire (*random choice*) pour la sélection d'une technique d'optimisation pour la recherche d'une solution à un niveau donné. Dans la troisième hyper-heuristique que nous appelons hyper heuristique stochastique, nous proposons une nouvelle stratégie de sélection. Le choix d'une technique d'optimisation est tributaire d'une probabilité P. Avec une probabilité P, le choix d'une technique d'optimisation se fait selon la stratégie de *choice function*. Avec une probabilité 1-P, la technique d'optimisation est choisie selon la stratégie aléatoire « random choice ».

Le présent mémoire se compose de 4 chapitres, dans le premier chapitre nous présenterons un état de l'art sur les enchères combinatoires : comment est modélisé le problème de la détermination du gagnant, avec une brève présentation des différents travaux antérieurs. Dans le chapitre 2, nous ferons une étude comparative entre les méthodes de recherche heuristiques, métaheuristiques et hyper-heuristiques. Les approches hyper-heuristiques que nous avons proposées pour la résolution du problème de la détermination du gagnant seront présentées dans le chapitre 3. Dans le chapitre 4 nous présenterons les résultats numériques des différentes approches proposées. Enfin, nous terminerons ce mémoire par une conclusion et quelques perspectives.

CHAPITRE I

LE PROBLÈME WDP

1. INTRODUCTION

Dans la vie de tous les jours, nous sommes amenés à vendre ou à acheter des biens ou des services. Cette pratique est diverse et peut être classée en quatre types **[3]**, Nous avons:

- un vendeur et un acheteur, ceci s'appelle aussi négociation acheteur/vendeur.
- Plusieurs acheteurs et un vendeur, qu'on peut appeler aussi enchère ou enchère simple, où le vendeur soumet plusieurs objets (ou services) à la vente et c'est à l'acheteur qui propose la meilleure offre que reviendra le droit d'acquérir ce bien.
- Un acheteur et plusieurs vendeurs, appelée aussi enchère inversée : cette fois ci le vendeur essaiera de convaincre l'acheteur d'acquérir ce qu'il vend en lui montrant les différents avantages par rapport aux autres vendeurs.
- Et enfin, Plusieurs acheteurs et plusieurs vendeurs, ceci est tout simplement appelé *marché*.

Notre travail se focalise sur les enchères simples. Dans ce qui suit, nous commencerons par une introduction aux enchères avec un bref historique, ainsi que quelques exemples d'enchères qui sont entrées dans l'histoire. Puis nous présenterons les différents types d'enchères, la description des enchères combinatoires ainsi que leur modélisation. Avant de conclure ce chapitre, nous parlerons des différentes approches déjà appliquées au problème du gagnant dans les enchères combinatoires.

2. INTRODUCTION SUR LES ENCHÈRES

Le mot "enchère" vient du latin *augere* qui signifie augmenter. Cette pratique de vente de bien au plus offrant ne date pas d'hier, mais remonte à des milliers d'années, où elle a connu des ventes très spectaculaires. On peut citer comme l'un des exemples de ces ventes historiques la vente de la couronne de l'Empire Romain aux enchères, et ceci après que la garde prétorienne eut assassiné l'empereur Romain Pertinax[1]. Cette vente s'est conclue à 25.000 drachmes (monnaie romaine) en faveur de *Didius Julianus*. Hérodote décrit le marché de mariage de Babylone, où la main des jeunes est mise aux enchères, et c'est le plus offrant qui remporte l'enchère[2]. Et nous pouvons aussi parler de la récente vente aux enchères du tableau « Le Cri » du peintre

[1] **http://www.noctes-gallicanae.fr/Rome/bas_empire.htm**
[2] **http://fr.wikipedia.org/wiki/Ench%C3%A8re**

norvégien Edvard Munch, adjugé le 02 mai 2012 à plus de 119 millions de dollars. Il y a eu aussi un grand nombre d'autres enchères d'œuvres d'art célèbres : On peut citer deux tableaux de Picasso qui ont atteint entre 104 et 106 millions de dollars, ou une sculpture d'Alberto Giacometti « L'homme qui marche » adjugée à 104.32 millions de dollars.

3. DIFFÈRENTS TYPES D'ENCHÈRES

3.1. Enchère de base

L'enchère de base consiste en la vente d'un objet (ou service) à plusieurs acheteurs, qui devront concourir pour obtenir l'objet qu'ils convoitent. Ce type d'enchère est un modèle classique dans la vente de biens, et a par le passé connu la vente d'un empire (celui de Rome), ainsi que de la vente de tableaux de grand peintres, on peut citer : Picasso, Van gogh,…

Il existe 4 types d'enchères de base qui sont présentées dans ce qui suit:

3.1.1. Enchère Anglaise

Appelé aussi *enchère ascendante*. C'est une vente par augmentation du prix de départ, ouverte ou publique, se terminant lorsque plus aucune offre ne surenchérit la dernière offre annoncée. Ainsi l'enchérisseur récupère son dû après avoir payé le montant proposé. Cette pratique favorise la compétition, pouvant même dépasser largement le prix estimé.

Un exemple concret est celui du tableau d'Edvard Munch, « Le Cri » qui était estimé à 80 millions de dollars, et a été vendu à 119 millions de dollars. C'est surtout avec une forte concurrence que le prix a atteint le record mondial.

Il y a une variante de l'enchère anglaise, où ce n'est pas aux enchérisseurs d'augmenter l'enchère, mais au commissaire-priseur (ou l'encanteur, qui est la personne qui préside la vente aux enchères) de hausser les enchères. Les enchérisseurs ne pouvant payer la somme atteinte, se retirent de la course, et c'est le dernier client qui reste qui se verra attribué l'objet convoité.

3.1.2. Enchère Hollandaise

Contrairement à l'enchère anglaise, qui est ascendante, l'enchère Hollandaise est descendante, c'est-à-dire, le commissaire-priseur fixe un plafond sur le prix de

l'objet mis en vente et le diminue progressivement jusqu'à ce qu'un enchérisseur se déclare prenant, et il devra verser la somme atteinte.

On peut dire que cette enchère comporte un avantage pour le vendeur, car celui-ci fixera un prix qui dépasse l'estimation de l'objet ; ainsi, quand le commissaire-priseur atteint la somme que l'un des enchérisseurs peut payer, s'il est intéressé par l'article mis en vente alors il se déclare preneur, car celui-ci n'a aucune information sur l'estimation que portent les autres acheteurs, et donc il ne peut pas tenter de laisser le prix descendre plus que cela, au risque qu'un autre enchérisseur ne se présente. Ce qui n'est pas le cas dans les enchères anglaises, où les prix augmentent progressivement contrôlés par les enchérisseurs ; ainsi, si les clients arrêtent d'enchérir, le dernier client ayant présenté son intérêt à l'article l'acquiert, et dans ce cas, le client peut ne pas atteindre le prix max qu'il s'est fixé **[3]**. Tout comme l'enchère anglaise, la hollandaise est ouverte ou publique.

3.1.3. Enchère à enveloppe scellée au premier prix

Dans ce type d'enchère, les acheteurs soumettent leurs offres au vendeur en cachant leurs offres aux autre acheteurs sans que leurs concurrent ne sachent le montant de leurs offres ; c'est pour cette raison qu'elle est appelée enchère à enveloppe scellée. L'acheteur ayant soumis l'offre la plus élevée remporte l'enchère, et devra payer la somme proposée. Le principal inconvénient de cette enchère, est la sous-évaluation de l'objet mis en enchère par les acheteurs, et ainsi la soumission d'offre inférieure au prix "mérité" pour cet article.

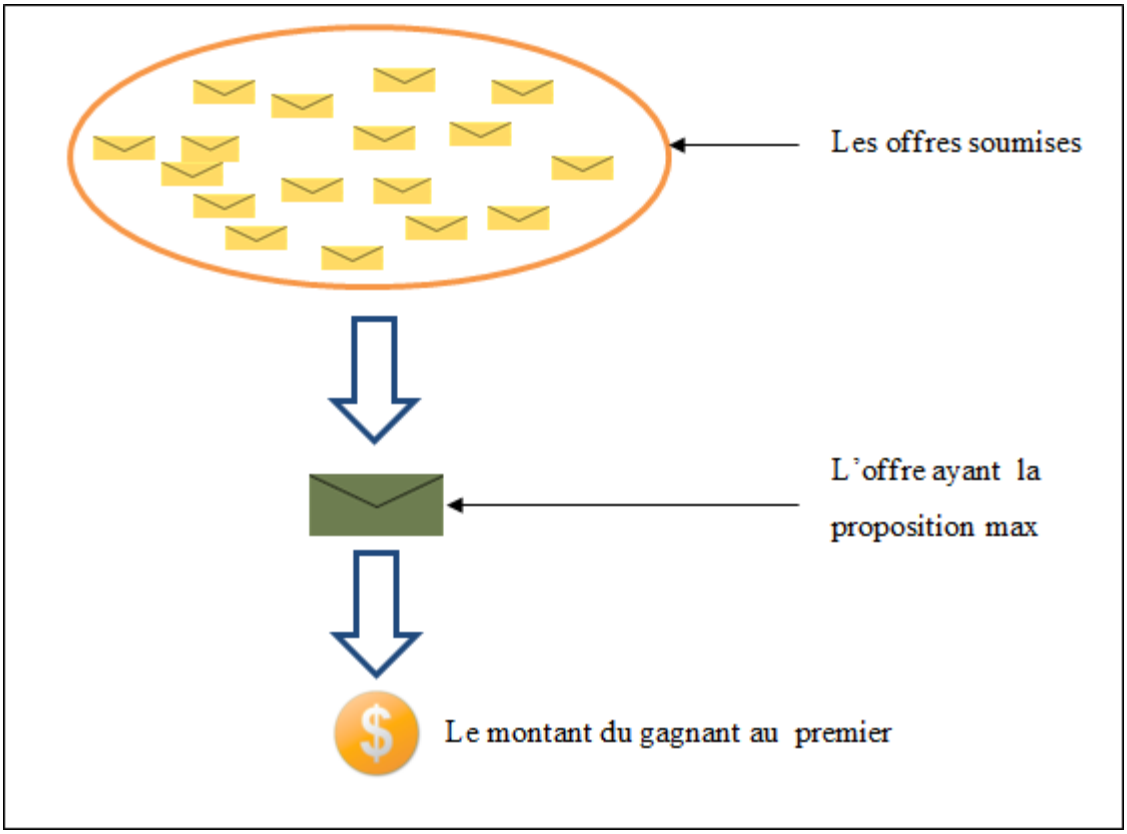

Figure 1. Exemple d'une enchère à enveloppe scellée au première prix

3.1.4. Enchère de Vickrey

Elle est aussi appelée *enchère scellée au deuxième prix*. Cette enchère ressemble à celle de l'enveloppe scellée au premier prix dans le déroulement de l'enchère, c'est-à-dire, dans la soumission des offres cachées aux acheteurs. La différence entre cette enchère et celle du premier prix, est que l'acheteur ayant soumis la plus grande offre, ne payera pas le montant proposé, mais le prix de la deuxième meilleure offre. Tout comme l'enchère à enveloppe scellée au premier prix, celle-ci présente un inconvénient où l'offre maximale proposée est inférieure à l'évaluation de l'objet mis en enchère, alors forcément la deuxième meilleure offre est inférieure à cette évaluation.

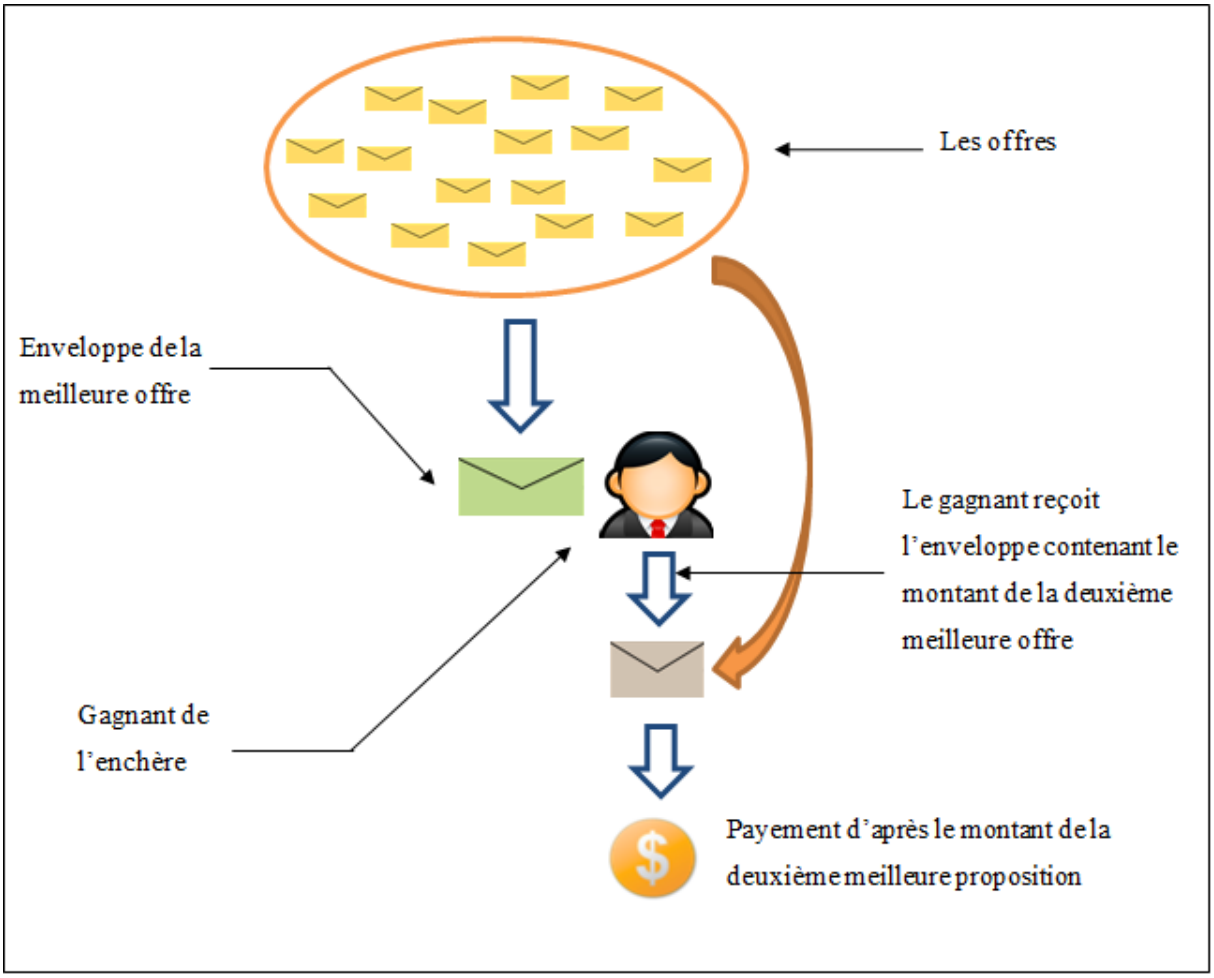

Figure 2. Exemple d'une enchère de Vickrey

3.2. Enchère multi-objet

Comme nous l'avons vu dans la partie 2.1 consacrée aux enchères mono-objet, il existe une autre manière de faire des enchères ; celle-ci concerne la vente de plusieurs objets. On peut avoir dans ce type d'enchère la soumission d'un ensemble d'objet considéré par l'acheteur comme complémentaire (on parle d'enchère combinatoire dans ce cas de figure), ou d'un ensemble d'objets identiques et indépendants les uns des autres **[3]**. Cette opération peut se faire de deux manières différentes, soit les objets font leurs apparition l'un après l'autre et ceci d'une manière séquentielle (comme cela se fait dans plusieurs ventes aux enchères où plusieurs objets d'antiquité, des timbres, les effets personnelle de personnalité nationale ou internationale, etc…), ou sinon d'une façon simultané où tous les objets sont vendus en même temps ; ce qui est le cas de la vente des licences radio dans les réseaux cellulaires par la Commission Fédérale de la communication américaine (FCC, *Federal Communication Commission*).

Dans ce mécanisme d'enchère multi-objets nous pouvons aussi citer les enchères multi-attributs **[4]**, où les ventes ne s'effectuent pas seulement sur le prix mais aussi sur certaines caractéristiques de l'objet exigé par l'acheteur. Ainsi, le client définit ces exigences sur l'objet qui l'intéresse, et c'est aux vendeurs de lui proposer ce qu'il exige.

Le type d'enchères multi-objets sur lesquelles porte notre travail est les enchères combinatoires.

4. DESCRIPTION DU PROBLÈME WDP

Le problème de la détermination du gagnant dans les enchères combinatoires (*winner determination problem*, WDP) ou PDG, est un problème d'optimisation NP-difficile qui peut se réduire vers le problème d'empaquetage pondéré d'ensemble (*weighted Set Packing Problem*), qui est connu pour être un problème NP-difficile **[37]**.

Il existe deux types de problème du PDG : mono-unité et multi-unités. Dans les enchères combinatoires mono-unité les objets ne possèdent pas d'exemplaire, ce qui veut dire que dans l'ensemble des objets de l'enchère, pour un objet x, il est le seul. Par contre, dans les enchères combinatoires multi-unités les objets peuvent posséder plusieurs exemplaires.

Dans ce qui suit, nous formalisons les concepts d'enchère combinatoire mono-unité et multi-unités.

4.1. Enchère Combinatoire mono-unité

Soit M = {1, 2,…, m} l'ensemble de m objets soumis à la vente aux enchères, et soit N= {1, 2, …, n} l'ensemble de n acheteurs (ou offres). Une offre j est représenté par le tuple $<S_j, P_j>$ où S_j correspond à l'ensemble des objets demandés par cette offre ($S_j \subseteq M$) avec le prix proposé P_j ($P_j \in \mathbb{R}^+$). Soit $[a_{ij}]_{m \times n}$ une matrice binaire où les lignes représentent les objets et les colonnes représentent les offres. Si un objet j est présent dans l'offre i, alors $a_{ij}=1$, sinon $a_{ij}=0$.

Soit x_j une variable binaire, indiquant la présence ou non d'une offre dans une solution donnée.

La résolution du problème WDP revient à trouver un ensemble d'offres permettant de maximiser le gain du vendeur tout en respectant une contrainte

importante. La fonction objectif ainsi que la contrainte du problème sont représentés comme suit :

$$max \sum_{j=0}^{n} P_j . x_j \qquad (1.1)$$

$$\text{s.t} \quad \sum_{j=0}^{n} a_{ij} . x_j \leq 1, \text{i} \in M \qquad (1.2)$$

avec: $x_j \in \{0,1\}$.

La contrainte (1.2) peut être résumée comme suit :

$$\forall \text{ i,j} \in X, \text{ i} \neq \text{j alors } S_i \cap S_j = \{\emptyset\} \qquad (1.3)$$

Avec X solution du problème.

Exemple :

Nous donnons un exemple pour illustrer les enchères combinatoires mono-unité, où nous avons un ensemble d'objet M = {1, 2, 3, 4, 5} soumis à la vente, et un groupe d'acheteurs souhaitant participer et où chacun ne soumet qu'une offre. Cette offre fera office d'identifiant. Nous prendrons dans cet exemple quatre acheteurs.

Le vendeur reçoit les offres des quatre acheteurs, ces offres sont l'ensemble des objets désirés S avec le montant proposé P. les quatre offres proposées sont décrites comme suit :

- Offre1 : < {1, 3, 5}, 230 DA >
- Offre2 : < {3, 5}, 250 DA >
- Offre3 : < {2, 4}, 170 DA >
- Offre4 : < {1, 2, 5}, 200 DA >

Des conflits existent entre Offre1 et Offre2, mais aussi entre Offre3 et Offre4, ce qui conduit à l'impossibilité de satisfaire toutes les offres en même temps.

Les offres 2 et 3, dont le profit des deux est égal à 420 DA, constituent la meilleure solution maximisant le profit du vendeur par rapport à la combinaison entre Offre1 et Offre4 ou si on acceptait les offres de manière isolée.

4.2. Enchère Combinatoire multi-unités [20]

La modélisation du problème du gagnant dans les enchères combinatoires dans le cas multi-unités est presque la même que celle du cas mono-unités. La différence réside dans le nombre d'exemplaires existant pour chaque objet, et donc nous avons :

Un ensemble M = {1, 2, …, m} d'objets en vente, un ensemble Q = {q_1, q_2,…, q_m} représentant le nombre d'exemplaires d'un objet présent dans l'enchère, où qi correspond à la quantité de l'objet i présent dans l'enchère. Une offre j est représentée par le tuple < S_j , P_j >, où P_j est le montant proposé, et S_j est l'ensemble d'élément a_{ij}, avec a_{ij} signifiant le nombre d'unité de l'objet i présent dans l'offre j.

La résolution du problème du PDG pour le cas multi-unités est la même que celle du cas mono-unité, c'est-à-dire, cela revient à trouver un ensemble X d'offres où il faut maximiser le profit du vendeur sous la contrainte de ne pas dépasser le nombre d'unités de chaque objet. Plus formellement, la contrainte est présentée comme suit :

$$\sum_{j=1}^{n} a_{ij}.x_j \leq q_i\ ,\ \ i = 1,2,\dots,m. \qquad avec\ \ x_j \in \{0,1\} \tag{1.4}$$

Exemple :

Nous donnons un exemple pour illustrer les enchères combinatoires multi-unités, où nous avons un ensemble d'objets M = {O1, O2, O3, O4} avec respectivement leurs nombre d'unité Q = {2, 2, 3, 4} soumis à la vente. Nous prendrons trois acheteurs ne soumettant qu'au maximum une offre numérotée, qui sera l'identifiant de l'acheteur l'ayant soumis.

Le vendeur reçoit les offres des quatre acheteurs, ces offres sont l'ensemble S des objets, avec pour chacun le nombre d'unités désiré, ainsi que le montant proposé P. pour un souci de clarté, nous n'utiliserons pas seulement la quantité d'objets utilisés mais aussi l'identifiant de l'objet dans l'ensemble S. Les trois offres proposées sont décrites comme suit :

- Offre1 : < {(O1, 2), (O3, 3), (O4, 1)}, 230 DA >
- Offre2 : < {(O1, 1), (O4, 1)}, 250 DA >
- Offre3 : < {(O2, 2), (O4, 1)}, 170 DA >

Dans ce type d'enchère, un conflit entre les offres est le fait de dépasser la quantité d'objets présente dans le stock. Des conflits existent entre Offre1 et Offre2, ce conflit est dû à la sur-demande de l'objet O1 si l'Offre1 et l'Offre2 sont

sélectionnés pour appartenir à la solution, car l'objet O1 existe en 2 exemplaires, et les deux offres 1 et 2 demandent respectivement 2 et 1 exemplaires ce qui donne 3 > 2.

Les offres 1 et 3, dont le profit des deux est égal à 400 DA, constituent la meilleure solution maximisant le profit du vendeur par rapport à la solution {Offre2, Offre3} ou si on prenait chaque offre seule. Dans notre travail nous nous intéressons aux enchères combinatoires mono-unité.

5. TRAVAUX ANTERIEURS AU PROBLÈME

Plusieurs travaux ont vu le jour pour la résolution du problème du gagnant dans les enchères combinatoires, nous pouvons citer deux grandes classes de méthodes appliquées au problème : les méthodes exactes (Branch & Bound, programmation dynamique, …) ainsi que les méthodes approchées (métaheuristiques). La première classe de résolution est efficace mais utilise de petites instances du problème par rapport aux méthodes approchées qui travaillent sur des instances avec des milliers d'offres et des milliers d'objets.

Parmi les méthodes exacte nous citons : Branch-on-Bids (BoB) **[45]**, Branch-on-Items (BoI) **[44]**, CABoB (*Combinatorial Auction BoB*) **[46]**, CASS (*Combinatorial Auction Structural Search*)**[21]**, CABRO (*Combinatorial Auction BRanch and Bound Optimizer*)**[37]**, une approche par programmation dynamique [**43**], CAMUS **[33]** une méthode qui traite le problème des enchères combinatoires multi-unités. Une méthode par programmation linéaire est proposée par [**38**], une méthode par la programmation en nombre entier **[1]**.

Pour ce qui est des méthodes approchées, nous pouvons citer : Casanova **[27]** qui est une méthode de recherche local, recherche local stochastique (SLS) **[6] [7]**, recherche taboue **[8]**, SAGII **[25]** qui est une méthode de recuit simulé hybridé avec un Branch-and-Bound, les algorithmes génétiques **[16]**, algorithme mémétique **[9]**, algorithme évolutionnaire différentiel **[5]**, une méthode de colonie de fourmis [**22**], PSO **[20]**.

5.1. CASS (Combinatorial Auction Structured Search)

CASS est un algorithme de Branch&Bound utilisant la stratégie de recherche en profondeur d'abord. Il procède d'abord par un prétraitement consistant à éliminer les offres dominant d'autre offres (c'est-à-dire, soit $g(b_i)$ l'ensemble des objets de l'offre

b_i, et soit $p(b_i)$ le prix proposé par cette offre, si pour une paire d'offre b_i et b_j, où $g(b_i) \subseteq g(b_j)$ avec $p(b_i) \geq p(b_j)$ alors, b_j domine b_i et est enlevé de la liste des offres potentielles).

CASS calcule une fonction $h(\pi)$ dans chaque nœud, qui donne une borne supérieure du revenu qui peut être obtenu à partir des objets restants qui ne sont pas encore exploré (c'est-à-dire, $G \backslash g(\pi)$ où G correspond à l'ensemble des objets du problème), si dans un nœud donné $p(\pi) + h(\pi) \leq p(\pi_{best})$ alors on élague le tronc, et on fait un retour arrière pour tester un autre chemin. Il utilise aussi la notion de "boites" (ou bins), celle-ci a pour but de réduire le nombre d'allocations infaisables. Cette technique commence par créer un nombre précis de boites qui correspond au nombre d'objets existants dans le problème, puis classe les objets entre eux, et mettra dans ces boites les offres ayant l'objet de classement le plus bas. Ce processus de scoring est réalisé grâce à la formule suivante :

$$score_i = numOffres_i / moyObjets_i$$

où $numOffres_i$ correspond au nombre d'offres contenant l'objet i, et $moyObjets_i$ correspond à la moyenne des objets contenus dans ces offres.

Et ainsi, au lieu d'essayer d'ajouter chaque offre dans la solution, on ajoute au plus une offre de chaque boite, comme toutes les offres présentes dans les boites sont en conflit, ce qui évite à l'algorithme de visiter toute les offres. Une autre fonction de tri est utilisé, cette fois ci c'est sur les offres dans les boites que le tri est effectué pour mettre en avant les offres qui donneront une éventuelle meilleure solution, cette fonction est définie comme suit :

$$score(b_j) = \frac{p(b_j)}{|g(b_j)|} + h(\pi \cup b_j)$$

La figure 3, montre un exemple de création de 5 boites correspondant au nombre d'objets existants dans le problème et la répartition des offres dans ces boites, en respectant le score le plus bas de leur objet.

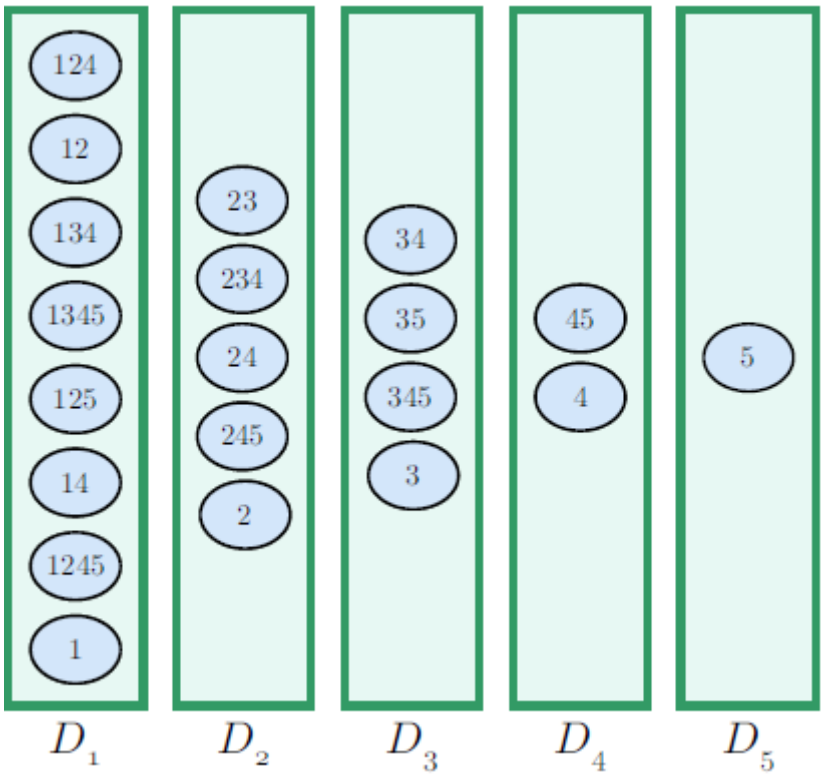

Figure 3. Partitionnement en boite (bins) [32]

Cette notion aide la fonction d'élagage à gagner du temps, car elle se basera sur les boites restantes et évitera les boites dont les objets de la solution leurs correspondent, exemple, si nous somme dans la boite Di, l'algorithme ne tiendra en compte que les boites $\{D_i,\ldots, D_m\}$.

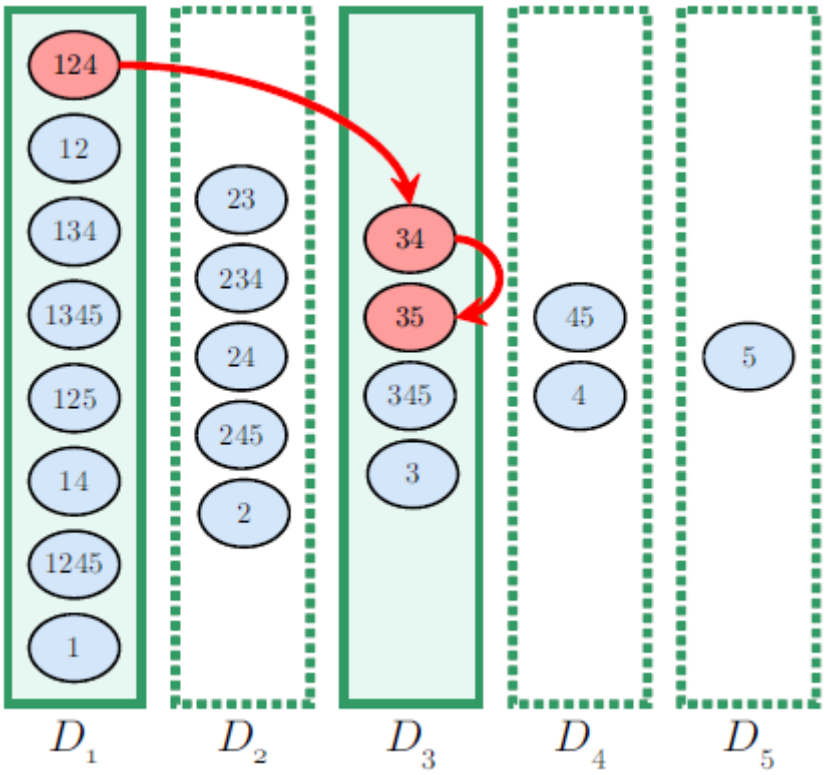

Figure 4. Exemple de boites évitées à cause de la présence de l'objet de la boite dans la solution partielle [32]

Une technique de "mise en cache" (caching) a été utilisée. Elle permet de serrer un peu plus la borne sup de la solution, ainsi pour éviter une surestimation du revenu. Il utilise une table de hachage sur l'ensemble des objets de la solution partielle atteinte ($g(\pi)$ et $c(g(\pi))$). L'algorithme commence à stocker la solution partielle à partir du moment où celle-ci ne contient pas plus de $|G|-k$ objets.

Pseudo-code : CASS

Prétraitement sur les offres dominantes.

Détermination de l'ordre des objets.

Distribution des offres dans les boites, d'après l'ordre de leurs objets.

i=1, $\pi = \{\}$.

Entrée Récursive :

$\pi = \pi \cup b_j$.

Si $(p(\pi) + c(g(\pi)) \leq p(\pi_{best}))$ alors retour arrière.

Si $(p(\pi) + h(\pi) \leq p(\pi_{best}))$ alors retour arrière.

Si aucune boite ne peut être explorée, alors mettre à jour π_{best} si nécessaire, puis retour arrière.

Mettre dans i l'objet ayant l'ordre le plus bas et absent dans π.

Trier les offres dynamiquement dans la boite i, et enlever les offres en conflit avec π.

Appel récursif.

$\pi = \pi \setminus b_j$.

Retourner l'allocation optimale : π_{best}

5.2. Recherche Locale Stochastique

La recherche locale stochastique (ou SLS, pour *Stochastic Local Search*) pour le problème du PDG a été présentée dans **[6][7]**, qui est un algorithme de recherche locale utilisant une fonction aléatoire pour la détermination du voisinage. L'algorithme commence par déterminer une solution initiale en utilisant la méthode de codage de clef aléatoire (ou *Random Key Encoding*), cette méthode attribue à chaque offre un poids entre 0 et 1, et sur la base de ces poids, elle prend les offres dont le poids est maximal. Ces offres ne doivent pas être en conflit avec celles (les offres) sélectionnées auparavant. La phase d'initialisation de la solution étant terminé, l'algorithme effectue d'une manière itérative l'insertion d'une offre qui ne se trouve pas dans la solution courante, tout en remédiant au problème d'infaisabilité d'une allocation (et ceci dans le cas où deux offres se partagent un ou plusieurs objets), cette phase de sélection d'une offre peut se produire selon deux critères :

Si la probabilité p < wp, alors une offre est choisie aléatoirement pour faire partie de la solution.

Sinon (alors p > wp) on choisit la meilleure offre pouvant améliorer la solution.

Dans cet algorithme, une solution est représentée par un vecteur V de taille $|V| \leq n$, ainsi chaque élément du vecteur V reçoit le numéro de l'offre sélectionnée. SLS utilise aussi un graphe de conflit indiquant les différentes offres en conflit, sachant que les sommets correspondent aux offres et les arcs entre les sommets donnent les objets en commun.

Afin de pouvoir juger la qualité d'une solution, une fonction d'évaluation est requise dans le but de la quantifier, celle-ci est définie comme suit :

$$Eval(V) = \sum_{i=1}^{l} p(V_i) \quad , \text{où } V_i \; offre \; présente \; dans \; la \; solution \; V \qquad (1.5)$$

L'algorithme s'arrête après *max_iter* itérations.

Algorithme 1 : Stochastic Local Search

Entrée : Instance du problème WDP, max_iter , wp.

Sortie : Allocation V.

Etablir le graphe de conflit a.

Sur la base de l'encodage de clef aléatoire, généré la solution initiale.

Pour I = 1 à max_iter **faire**

r <= nombre aléatoire entre 0 et 1.

Si (r < wp) alors

Offre <= une offre qui sera choisie aléatoirement.

Sinon

Offre <= choisir la meilleure offre (avec la fonction de voisinage)

finSi.

V = V ∪ {Offre}.

En se basant sur le graphe de conflit (à l'aide de la matrice a), enlever tous les conflits dans V.

Fait ;

Retourner l'allocation V.

5.3. Recherche taboue

La recherche taboue est une métaheuristique à voisinage possédant une mémoire dite *liste taboue*, qui permet d'éviter les solutions déjà visitées. L'approche a été proposée par Fred Glover **[24]**.

La recherche taboue appliquée au problème du PDG **[4] [8]** utilise la même représentation d'une solution que celle du SLS vu précédemment, qui est un vecteur de longueur variable, ainsi que d'une même fonction d'évaluation.

Cette recherche commence par générer, comme pour le SLS, une solution initiale en utilisant l'encodage de clef aléatoire. La recherche du voisinage d'une solution peut se faire à l'aide de deux opérateurs de mouvements : Sur-l'offre (*on-bid*) et Sur-l'objet (*on-item*), ces deux mouvements choisissent la meilleure offre insatisfaite (qui n'est pas présente dans la solution) et ainsi enlever toute les offres en conflits dans l'allocation.

Une fois l'offre ajoutée à l'allocation, celle-ci reçoit le statu taboue pour λ itération, afin d'éviter les optimum locaux, après cela l'offre sera libérée. Un critère intéressant a été utilisé, il donne la possibilité d'accepter un mouvement venant d'une offre taboue, et ceci à la seule condition qu'elle puisse améliorer la solution, ce critère est appelé critère d'*aspiration*.

Dans le cas où aucune amélioration n'est atteinte après un certain nombre d'itération d, alors un processus de '*diversification*' est utilisé pour explorer d'autres régions et ainsi sortir de cette stagnation dans la recherche. Apres max_iter itérations la recherche s'arrête.

Algorithme 2 : Tabu Search

Entrée : Une instance du PDG, max_iter, d, λ.

Sortie : l'allocation V.

TL = Ø. // TL est la liste taboue.

V_{best} = Ø.

F_{best} = 0.

iter = 1.

Générer le graphe de conflit.

Sur la base de l'encodage de clef aléatoire, générer la solution initiale. V reçoit cette solution.

V_{best} = V.

F_{best} = Eval(V).

Tantque (iter ≤ max_iter) faire

Générer des voisins de V à l'aide des opérateurs de mouvement, et choisir la meilleure.

Appliquer le critère d'aspiration si la meilleure solution est généré par l'insertion d'une offre taboue dans l'allocation.

V=V ∪ { meilleure offre }.

Enlever les conflits existants dans V.

F = Eval(V).

Si (F > F_{best}) **alors**

V_{best} = V.

F_{best} = F.

Iter_best = iter.

FinSi.

Si (iter - iter_best ≤ d) **alors**

Phase de diversification.

FinSi.

Fait.

Retourner la meilleure solution.

5.4. Algorithme mémétique

L'algorithme mémétique s'inspire des algorithmes génétiques pour ce qui est des opérations de sélection, croisement et mutation, et rajoute à cela une méthode de recherche locale pour améliorer l'adaptation d'un individu**[35]**.

L'approche développée par **[9]**, tout comme la recherche taboue ou SLS, utilise l'encodage de clef aléatoire pour générer la population initiale. Pour ce qui s'agit du processus de reproduction (i.e., création de nouvelles solutions) pour la nouvelle génération, l'algorithme travaille sur un ensemble C, composé de deux sous-ensembles : le premier contient les |C1| meilleurs individus évalués grâce à la fonction d'évaluation (1.5), ainsi que de |C| - |C1| éléments dits les plus diversifiés pris de l'ensemble P - C1 ; ces derniers éléments sont sélectionnés grâce à une fonction dite de similarité, qui calcule la similarité entre deux éléments, cette similarité se traduit par le nombre de gènes en commun entre un individu X et un autre individu Y. Cette fonction est définie comme suit :

$$Sm_C(X)=max_{y\in C}(Sm(X,Y)) \qquad (1.6)$$

Une opération de croisement est effectuée sur les individus de C, produisant ainsi une progéniture engendrée par deux parents (noté Parent1 et Parent2), cette opération croise les gènes des Parents en commençant par Parent1 puis il passe vers celui du Parent2 en évitant bien évidement les conflits entre les gènes.

Ce qui résulte de l'opération de croisement passera par la méthode de recherche locale stochastique (SLS) afin d'améliorer l'individu engendré par ses parents. Si l'individu possède une bonne qualité après son évaluation, alors il appartiendra à l'ensemble C1, sinon, on vérifie si celui-ci améliore la diversité de l'ensemble restant.

L'algorithme s'arrête après *max_iter* itérations.

Algorithme 3 : Memetic Algorithm.

Entrée : une instance du PDG, wp, max_iter.

Sortie : l'allocation V.

Générer le graphe de conflit des offres.

Générer la population initiale avec l'encodage de clef aléatoire.

Générer l'ensemble de solution C à partir de la population P.

Pour iter = 1 à *max_iter* **faire Si**

Répéter

- Sélectionner deux individus dans C.
- Appliquer l'opération de croisement sur ces individus sélectionnés, on obtient V.
- Appliquer la recherche SLS sur V.
- Evaluer V et calculer sa diversification par rapport à la population.

Si (V améliore la qualité de C) **alors**

Ajouter V dans C1, et enlever la plus mauvaise solution dans C.

Sinon (V améliore la diversité de C) **alors**

Ajouter V dans C2, et enlever l'individu qui a la valeur de similarité la plus grande de C.

FinSi.

Jusqu'à ce que tous les parents soit traités.

Fait.

Retourner la meilleure solution.

5.5. Algorithme d'Evolution Différentielle

L'évolution différentielle est une approche évolutionnaire à population proposée par Storn et Price **[50]**. Cette approche a été utilisée pour la résolution de certains problèmes d'optimisation, dont le problème WDP.

L'algorithme à évolution différentielle proposé par **[9]** pour la résolution du PDG est décrit comme suit :

L'algorithme commence par générer une population initiale à l'aide de l'encodage à clef aléatoire. à chaque individu W de la population P sera associé un vecteur appelé vecteur mutant, noté U, afin qu'il puisse participer à l'opération de reproduction d'une nouvelle solution avec W. Ce vecteur mutant résulte de l'opération de génération suivante :

$$U = X_{best} + F.(Y-Z)$$

Où Y et Z sont deux individus tirés aléatoirement de la population pour contribuer à la génération de U ; X_{best} est le meilleur individu de la population, et enfin F est un scalaire strictement positif. L'individu engendré par le croisement de U et W est noté par V, celui-ci passera par la recherche locale SLS afin d'essayer de l'améliorer en vue de la phase d'intensification de l'algorithme.

Le meilleur entre le parent W et son enfant V, sera sélectionné pour faire partie de la nouvelle génération.

Algorithme 4 : D.E Algorithm

Entrée : instance du PDG, wp, *max_iter*.

Sorite : Une allocation maximisant le revenu du vendeur.

Générer le graphe de conflit.

Générer la population initiale à l'aide de l'encodage à clef aléatoire.

Pour iter = 1 à *max_iter* **faire**

Pour chaque individu X de P **faire**

- Calcule du vecteur mutant U
- Croisement entre X et U, on obtient V.
- Enlever les conflits entre les offres de V.
- Appliqué SLS sur V, on obtient V'.
- Choisir le meilleur individu entre X et V'.

Fait.

Fait.

Retourner la meilleure solution.

6. CONCLUSION

Le problème du gagnant dans les enchères combinatoires a vu la présentation et le développement de plusieurs approches dans le but de sa résolution. La limite des méthodes exactes dans la résolution du problème à plus grande échelle, a contribué à l'utilisation des méthodes approchées à ces problèmes.

En conséquence, notre travail s'est porté sur l'utilisation d'un ensemble de ces approches de résolution non-exhaustive (approchée) pour un travail coordonné dans le

but de résoudre le problème du PDG avec une méthode (Hyper-Heuristique) qui servira de chef pour ces méthodes.

Le chapitre qui suit donnera une étude comparative entre les méthodes approchées : Heuristiques, Métaheuristiques, et hyper-heuristiques.

CHAPITRE II

Étude comparative entre les heuristiques, métaheuristiques et hyper-heuristiques

1. INTRODUCTION

Souvent les techniciens, ingénieurs, chercheurs… sont confrontés à des problèmes d'optimisation combinatoire, où dans la plupart des cas, le résultat de bonne qualité est important et déterminant.

Un problème d'optimisation combinatoire, est un problème dont on cherche son optimum (solution minimale, ou maximale) à l'aide d'une fonction objective (ou plusieurs, et dans ce cas nous sommes dans un problème d'optimisation multi-objectifs) et qui vérifie les contraintes fixées du problème.

Ces problèmes d'optimisation combinatoire ont été traités à l'aide de méthodes exhaustives ou méthodes exactes, c'est-à-dire, des méthodes qui donnent la solution optimale après avoir exploré tout l'espace de recherche, en évaluant les solutions admissibles, et à la fin du processus, elles retournent la meilleure solution du problème. Mais ces méthodes deviennent lourdes dès que l'instance d'un problème devient grande, et ceci nous conduit vers une explosion combinatoire du nombre de solutions à explorer, ce qui est en pratique, un handicap pour le processus de recherche. C'est ainsi que de nouvelle méthodes sont apparues, qui ne donnent certes pas la solution optimale, mais plutôt une solution proche de l'optimale dans un temps raisonnable. Ces méthodes sont dites *méthodes approchées*.

Puisque actuellement, aucun algorithme de résolution ne peut donner, dans un temps polynomial, la solution optimale pour des instances de grande de taille (jusqu'à ce que l'on trouve P = NP, ou P ≠ NP) **[52]**, le regard des chercheurs s'est tourné vers le développement de ces méthodes approchées, afin de les rendre plus robustes et plus performantes : qu'elles puissent nous assurer une solution de bonne qualité qui soit de plus en plus proche de l'optimum. Un exemple qui illustre de manière convaincante la nécessité de travailler avec des méthodes approchées est cité dans **[41]**, où l'auteur nous donne un petit aperçu du temps que peut prendre un algorithme exact pour donner la solution optimale au problème du voyageur de commerce sur une matrice carrée de taille 60*60, et qui est égale à 10^{71} secondes, en précisant que l'âge de l'univers est de $\mathbf{10^{17}}$ **secondes**. C'est dire l'importance de passer vers des méthodes de résolution approchées.

Dans ce chapitre nous ferons une brève présentation de chaque approche de résolutions approchées, avec quelques exemples, ce qui nous mènera à une étude

comparative entre ces 3 approches sur différents axes, et pour finir, une conclusion qui résumera ce que nous aurons vu dans ce chapitre 2.

2. OPTIMISATION COMBINATOIRE

L'optimisation combinatoire est une branche de l'optimisation, elle occupe une place très importante en recherche opérationnelle, en mathématique discrète et en informatique **[26]**. Les problèmes dits d'optimisation combinatoire, comme cela a été présenté dans l'introduction de ce chapitre, consistent à trouver pour des problèmes à espace de solutions dénombrables, une solution qui soit son optimum et puisse respecter ces contraintes. Plus formellement, un problème d'optimisation combinatoire est défini comme suit :

Soit S un ensemble fini (mais qui peut être très grand) de solutions du problème appelé aussi espace de recherche ou espace de solution, et soit X, $X \subseteq S$ l'ensemble de solutions admissibles du problème. Et soit $f : S \longrightarrow \mathbb{R}$, appelé fonction objectif, qui associe à chaque élément de S un réel représentant sa qualité. Nous cherchons à trouver une solution $x_{best} \in X$, dans laquelle x_{best} est l'optimum du problème, ou :

$$f(x_{best}) = \mathrm{OPT}_{x \in X} \; f(x), \quad \text{avec OPT} = \{\min, \max\}.$$

3. LES HEURISTIQUES

Le mot heuristique ou *euristique* vient du grec *heuriskein*, qui signifie l'art d'inventer, de faire des découvertes[3]. Nous pouvons dire que les heuristiques sont « *tous les outils intellectuels, tous les procédés et plus généralement toutes les démarches favorisant la découverte ou l'invention* » **[34]**, et donc c'est une collection d'informations du domaine traité, permettant de guider le processus de recherche **[51][19][36]**.

Les heuristiques sont utilisées dans plusieurs méthodes de résolution (exacte ou approchée), elles permettent à ces dernières d'effectuer des mouvements afin d'améliorer les solutions courantes.

Quand on parle d'algorithme de recherche heuristique, on parle de méthode approchée utilisant les connaissances propres au problème traité. Ce qui veut dire que cette méthode est spécifique au problème, offrant une solution proche de l'optimale

[3] **http://fr.wikipedia.org/wiki/Heuristique**

d'assez bonne qualité avec une exécution plus rapide que les méthodes exhaustives pour des instances de problèmes de grande taille **[19]**.

Nous pouvons illustrer cela par un exemple, celui du problème de placement d'antenne dans les réseaux cellulaires, où l'opérateur doit trouver un moyen de distribuer les antennes, assurant la fonction d'émetteur/récepteur, sur une zone géographique afin de permettre à ses clients de profiter pleinement des services proposés. Notons que les émetteurs/récepteurs coutent cher aux opérateurs, et surtout si nous ajoutons leurs maintenances cela pourrait être couteux pour eux. C'est dans ce sens que l'objectif du problème de placement d'antenne est de minimiser les antennes déployées dans la zone tout en assurant une couverture maximale de cette zone.

Figure 5. Exemple de réseau cellulaire

Après analyse des données recueillies sur la zone à couvrir (exemple : densité de la population,…), le concepteur peut éventuellement proposer un algorithme heuristique, qui sélectionne des antennes et les places dans les régions à forte densité, jusqu'à atteindre des zone moins fréquentées. Celle-ci est une approche que le concepteur pourra adopter, voire ajouter d'autres paramètres, d'autres données, ou sinon passer à une autre méthode.

4. LES MÉTAHEURISTIQUES

4.1. Introduction

Le terme métaheuristique a été utilisé pour la première fois par Glover en 1986, après avoir présenté la recherche taboue **[23]**.

Les métaheuristiques se veulent génériques, des méthodes totalement indépendantes des problèmes à résoudre, des méthodes qui ont changées la façon de résoudre des problèmes difficiles en s'inspirant parfois de phénomènes naturels. Ces méthodes sont le contraire des algorithmes de recherche heuristique qui sont liés au problème traité **[36][23]**.

Si à partir d'une méthode de recherche heuristique nous pouvions extraire un schéma général, de tel manière que celui-ci ne soit plus lié au problème traité, alors cette méthode deviens une méthode générale avec un haut niveau d'abstraction lui permettant d'aborder n'importe quel problème, et de cette façon elle devient une métaheuristique **[34]**.

Un autre point important des métaheuristiques est de parcourir l'espace de recherche du problème afin de trouver son optimum global. Le hic dans tout cela est qu'il existe des pièges appelés *optimum locaux*, qu'il faut absolument éviter si nous voulons atteindre l'optimum global, et ceci peut se faire grâce à un bon paramétrage (s'il existe), des opérations de génération de solution efficace,… ect.

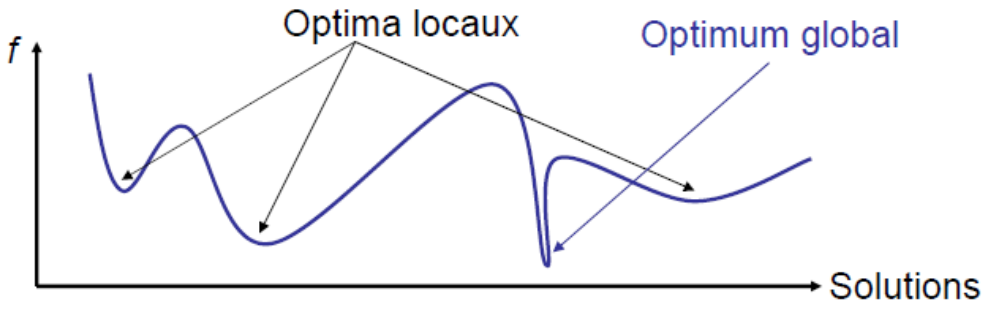

Figure 6. Exemple d'optimum local et global pour un problème de minimisation [4]

En parlant de paramétrage, celle-ci est à la fois un avantage et un inconvénient. L'avantage est qu'il permet à l'algorithme de mieux parcourir l'espace de recherche et ainsi il pourra aboutir à la solution optimale, mais pour arriver à un bon paramétrage il faut le déterminer, et pour le faire il faut expérimenter les différentes valeurs qu'il(s) peut (peuvent) prendre, ce qui représente leurs inconvénients.

Les métaheuristiques ont un autre point commun, qui est la condition d'arrêt qui est fixé, qui contrairement aux méthodes exactes qui s'arrêtent après avoir trouvé la

[4] Arnaud Liefooghe, cour : « ***Introduction aux métaheuristiques*** »

solution optimale, celle-ci peuvent avoir des conditions d'arrêt par rapport au nombre d'itérations de l'algorithme, le temps d'exécution,…

4.2. Maintien de la balance entre Diversification et Intensification

La *diversification* (ou l'*exploration*) et l'*intensification* (ou l'*exploitation*) sont deux concepts importants dans le processus de recherche dans une métaheuristique, où la diversification vise à explorer d'autres zones de l'espace de recherche afin de tomber sur une zone pouvant contenir l'optimum global, tandis que l'intensification, comme son nom l'indique, intensifie la recherche sur la zone en cours afin d'améliorer de plus en plus la solution courante.

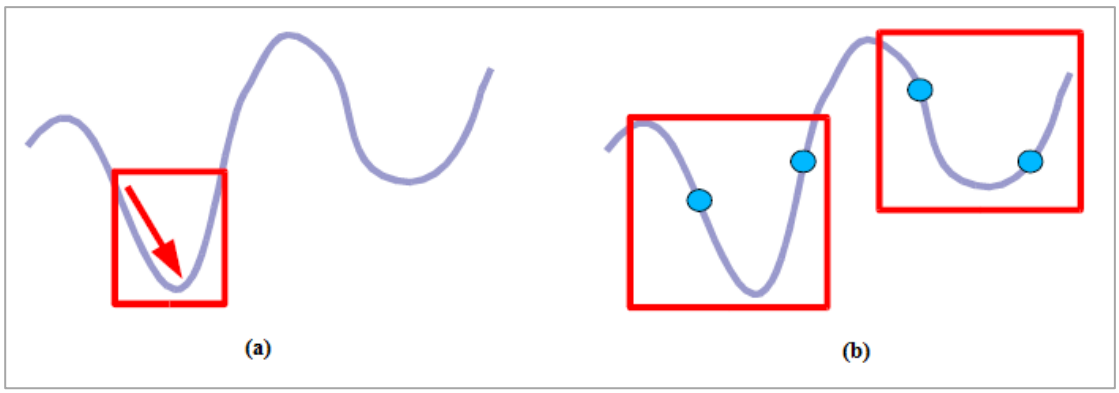

Figure 7. Principe d'intensification (a), et de diversification (b) [19]

L'équilibre entre ces deux concepts est très important, si notre métaheuristique penche plus vers l'intensification alors on risque de converger vers un optimum local sans pour autant assurer une sortie, et si la balance penche plutôt vers la diversification, alors on risque de parcourir plusieurs zones de l'espace de recherche sans pour autant en exploiter l'une d'elle.

4.3. Classification des métaheuristiques

Plusieurs classifications des métaheuristiques existent **[48]**. Cette diversité est le fruit de différents points de vue considérés, et ceci en regroupant leurs caractéristiques en commun (présence ou non d'une mémoire, inspirées de phénomènes naturels ou non,…).

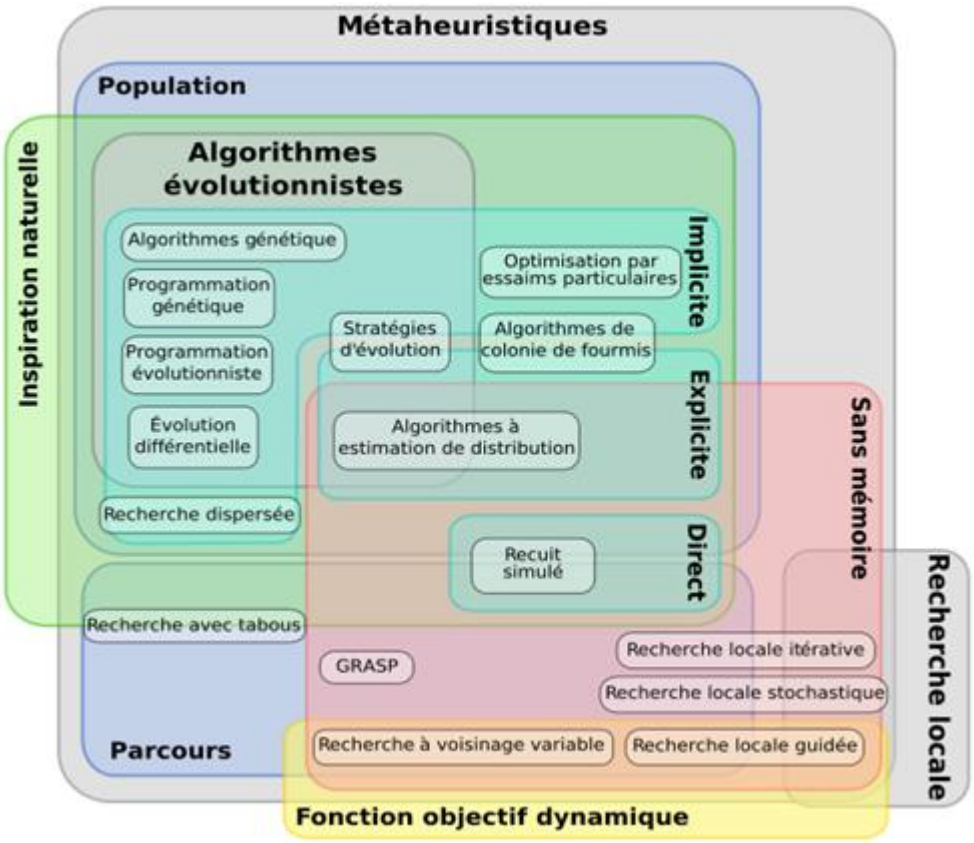

Figure 8. Classification de métaheuristique sous plusieurs angles [36]

La classification la plus citée est celle du nombre de solutions traitées en même temps. Nous avons deux grandes classes : *les métaheuristques à base de solution unique*, celles-ci travaillent sur une seule solution à la fois et tentent de chercher d'autres solutions dans son voisinage. Ces métaheuristiques choisissent la prochaine solution d'après une certaine stratégie propre à elles. L'autre classe est : *les metaheuristiques à base de population*, celles-ci travaillent avec un ensemble de solutions simultanément. Cette ensemble de solutions est appelé *population*. Cette classe de méthodes tente de faire évoluer cette population à chaque génération afin d'essayer d'atteindre l'optimum global.

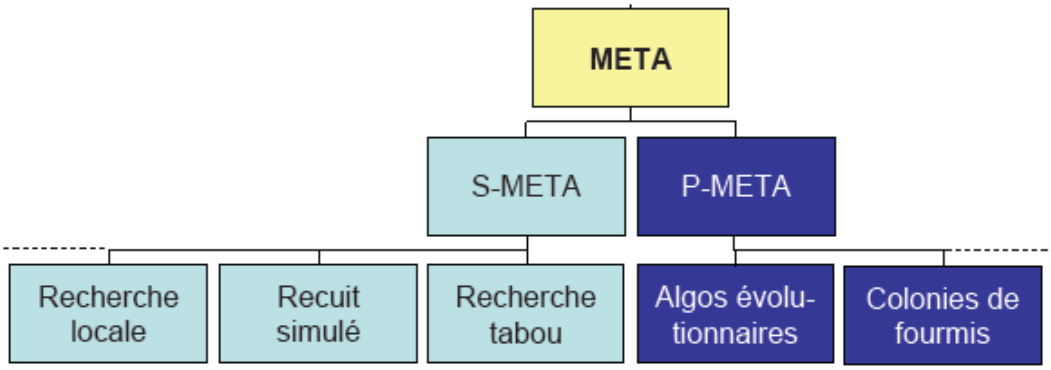

Figure 9. métaheuristique basée population et métaheuristique basée solution unique

Dans ce domaine de l'optimisation d'autres modèles peuvent apparaitre, et ceci grâce au développement d'autres disciplines (médecine, mécanique,…ect), car c'est en s'inspirant des modèles réels, que des modèles virtuels seront créés pour apporter une nouvelle vision de la résolution des problèmes.

5. LES HYPER-HEURISTIQUES

5.1. Introduction

Il existe d'autres méthodes approchées de plus haut niveau, indépendantes du problème **[30]**, faisant interagir plusieurs métaheuristiques et/ou algorithmes heuristiques spécifiques au problème à traiter, ces méthodes ont pour nom *Hyper-heuristiques.*

Ce terme a été employé en 2000 [**11**], et il a été décrit comme le concept « d'*heuristique qui choisit des heuristiques* ». Ces méthodes ont été utilisées dans plusieurs problèmes dont : le problème d'affectation de fréquences dans les réseaux cellulaires (FAP)[**29**], le problème de l'emploi du temps des examens (*Examination Timetabling Problem*)**[39]**[**14**]**[47][28][40]**, problème de planification **[17]**,…

Ces méthodes n'effectuent pas directement la recherche sur l'espace de solutions, mais indirectement à travers les méthodes de résolution du problème (métaheuristique et/ou algorithme de recherche heuristique), et donc travaille sur l'espace de méthodes [**12**][**49**][**42][39]**.

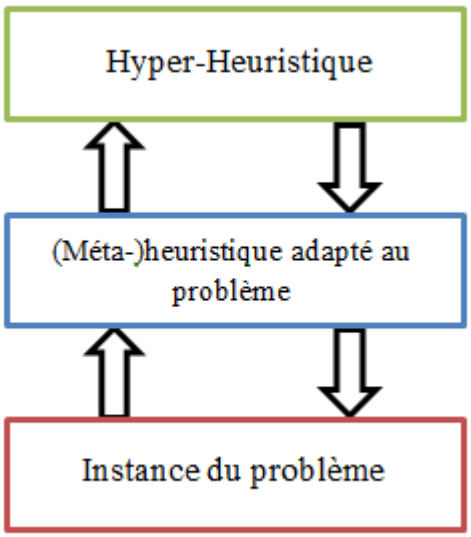

Figure 10. Différent niveau d'interaction dans une plateforme hyper-heuristique.

Ce qui nous conduit à dire que les hyper-heuristiques tentent de trouver, à partir d'un ensemble d'(méta-)heuristique, la bonne méthode dans une situation particulière, au lieu d'essayer de résoudre le problème directement**[49]**.

L'hyper-heuristique peut travailler avec des informations indépendantes du problème, ces informations sont des indicateurs de performance, par exemple : temps CPU de chaque exécution, la qualité d'une solution trouvée, …etc. Ce qui lui permet de décider à quelle heuristique faire appel pour la prochaine exécution **[49][17]**.

Comme différentes métaheuristiques ainsi que les méthodes de recherche heuristique possèdent différentes forces et faiblesses, les hyper-heuristiques tentent de combiner ces méthodes afin d'arriver à compenser les faiblesses de certains par la force des autres **[49]**.

Une nouvelle définition des hyper-heuristiques est apparue après que la programmation génétique eut été utilisée comme hyper-heuristique **[11]**. Ainsi l'auteur définit l'hyper-heuristique comme étant « *une méthode automatique pour la génération ou la sélection de méthode heuristique (et/ou métaheuristique) pour la résolution de problèmes difficiles* ».

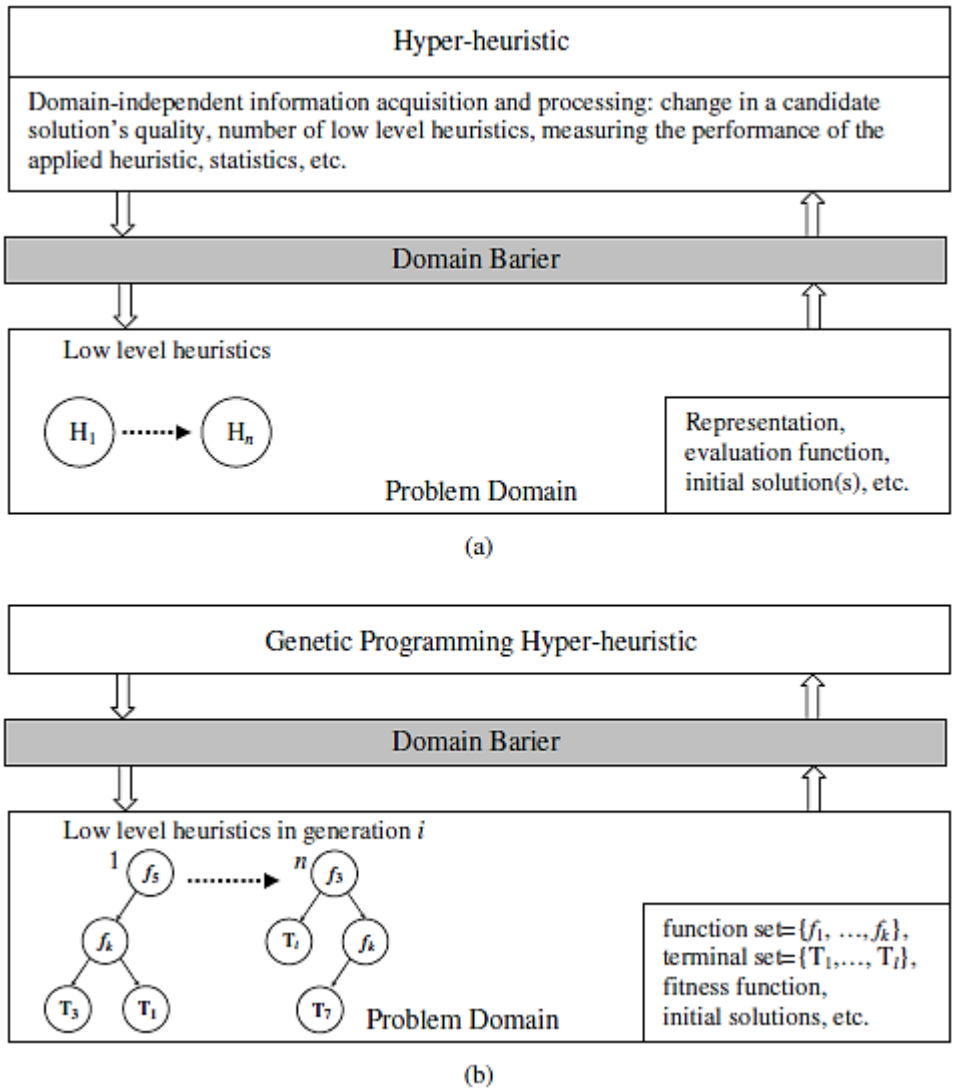

Figure 11. Hyper-heuristique de sélection (a) vs Hyper-heuristique de génération (b) [13]

5.2. Classification des Hyper-Heuristiques

On peut voir une hyper-heuristique comme étant une méthode de haut niveau manipulant un ensemble de métaheuristiques et/ou de méthodes de recherche heuristiques adaptées au problème traiter ou des composants. Cet ensemble est appelé ensemble d'*heuristique de bas niveau*, dans le but de produire des solutions résolvant de manière efficace ces problèmes.

Les auteurs de [**11**] présentent les différentes classifications faites sur les hyper-heuristiques, à commencer par **[49]** qui présente une classification selon que l'hyper-heuristique : (i) travaille avec une méthode d'apprentissage, ou (ii) sans apprentissage.

Une autre classification des hyper-heuristiques est présentée dans **[2]** où l'auteur considère qu'il existe deux classes : (i) les hyper-heuristiques manipulant des

heuristiques de bas niveau constructive, ou (ii) les hyper-heuristiques travaillant avec des méthodes pérturbatives. Pour ce qui est de la première classe, elle démarre à partir d'une solution vide et construit petit à petit une solution complète, pour la deuxième classe, elle démarre à partir d'une solution initiale complète et tente à chaque étape de l'améliorée.

[15] présente 4 classes d'hyper-heuristique : (i) les hyper-heuristiques choisissant une heuristique de bas niveau aléatoirement, (ii) les hyper-heuristiques qui utilisent des mécanismes d'apprentissage afin de choisir les heuristiques de bas niveau, (iii) des métaheuristiques comme hyper-heuristique (exemple : Recherche Taboue, Recuit Simulé, ...), et enfin (iv) les hyper-heuristiques gloutonne (*Greedy hyper-heuristics*).

Dans [**11**], les auteurs se basent sur les précédentes classifications afin de présenté une classification plus générale, d'après deux dimensions qui sont : (i) la nature de l'espace de recherche heuristique, et (ii) l'utilisation ou non de technique d'apprentissage.

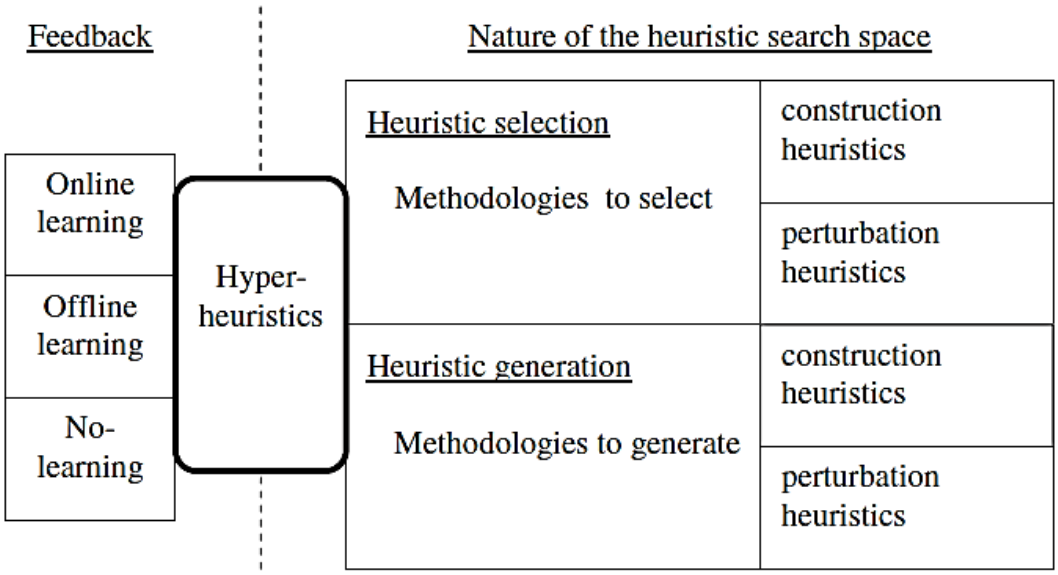

Figure 12. Classification des hyper-heuristiques [12]

Pour la première dimension nous avons :

- **Heuristique de sélection** (*heuristic selection*): qui est une méthode qui choisit une heuristique de bas niveau à partir d'une base.
- **Heuristique de génération** (*heuristic generation*) **[13]**: une méthode qui génère de nouvelles heuristiques à partir de composants existants.

L'autre aspect de la classification est l'apprentissage, si l'heuristique de haut niveau n'utilise aucune information renvoyée par les heuristiques de bas niveau, alors on dit qu'elle est non-apprenante, sinon il existe deux types d'apprentissage utilisés :

- **L'apprentissage en-ligne** : l'algorithme apprend en même temps qu'il procède à la résolution du problème.
- **L'apprentissage hors-ligne** : permet d'entrainer d'abord le programme avec des exemples, avant d'attaquer la résolution d'un problème.

En prenant un concept de chaque dimension, nous obtenant une hyper-heuristique.

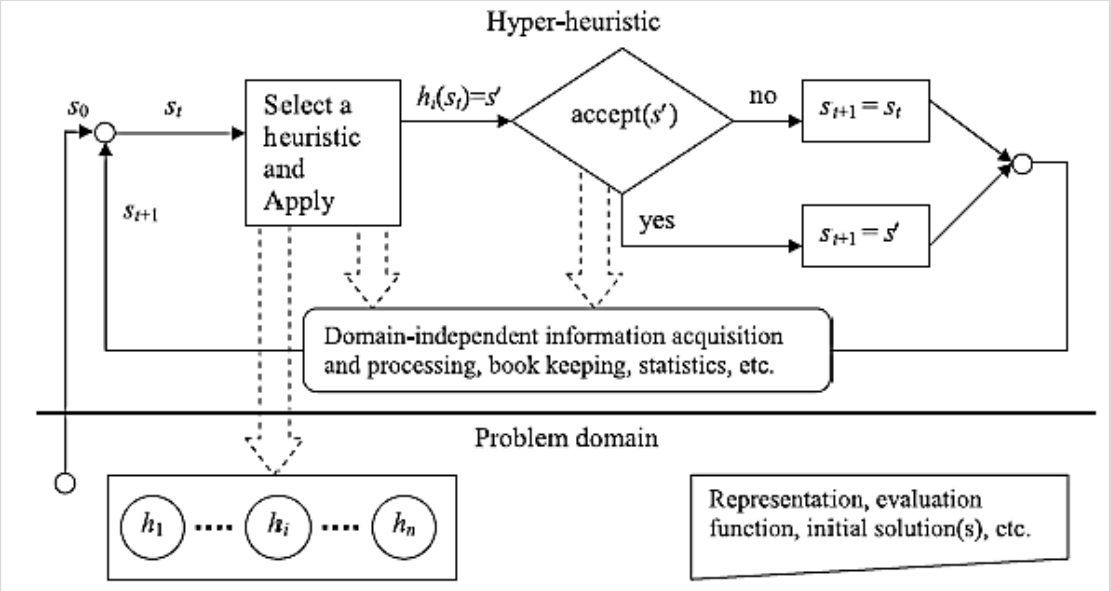

Figure 13. Hyper-heuristique de sélection d'heuristique de bas niveau perturbatrice [14]

[**11**], [**10**] et [**39**] décomposent l'hyper-heuristique perturbatrice en deux composantes principales :

- **L'heuristique de sélection** : qui effectue la tache de sélection d'heuristique de bas niveau, en s'appuyant ou non sur des mécanismes d'apprentissage.
- **Stratégie d'acceptation de la solution**: qui permet de donner le degré d'acceptation d'une solution renvoyer par une heuristique de bas niveau.

Dans ce type d'hyper-heuristique son appellation est la composition de la méthode de sélection heuristique avec la stratégie d'acceptation (exemple : *Choice Function – Only Improvement Hyper-heuristique*).

6. Tableau comparatif

Dans cette partie nous présentons un tableau comparatif (Tableau 1) entre les méthodes heuristiques, les métaheuristiques et les hyper-heuristiques comme résumé de ces trois méthodes de recherche, pour ce faire une étude s'est faite sur les critères de comparaison suivants :

- **L'entité(s) manipulée(s) :** Indique l'entité à laquelle la méthode de recherche effectue ses opérations, et ainsi l'entité qui est directement manipulée.
- **Opérateur(s) utilisé(s) :** Précise quels sont les opérateurs utilisés par ces méthodes de recherche.
- **Principe de diversification :** Indique si la méthode utilise ou non le principe de diversification dans son processus de recherche.
- **Principe d'intensification :** Indique si la méthode utilise ou non le principe d'intensification dans son processus de recherche.
- **Dépendance au problème :** comme son nom l'indique, donne le lien qui puisse exister entre la méthode et le problème.
- **Niveau d'abstraction :** Précise de manière générale le niveau d'abstraction de la méthode de recherche dans sa conception.
- **L'espace de recherche :** Précise l'espace sur lequel la méthode travaille.

Tableau 1. Tableau comparatif entre les méthodes heuristiques, métaheuristiques et hyper-heuristiques

	Méthode heuristique	**Métaheuristique**	**Hyper-heuristique**
Entité(s) manipulée(s)	Solution unique	Solution unique/ population de solution	Un ensemble de méthodes de recherche
Opérateur(s) utilisé(s)	Des mouvements dépendants du problème	Tout dépend de la métaheuristique, exemple : - Algorithme génétique : Sélection, Croisement, Mutation. - Recherche taboue : Mouvement dans le voisinage et une insertion dans la liste taboue. - …etc.	Tout dépend de l'hyper-heuristique, exemple : - Choice function : étude des performances des méthodes et choisir celle qui possède le rang max. - Hyperheuristique taboue : mise dans la liste taboue d'une méthode pendant k itérations. - … etc.
Principe de diversification	Absence	Présence	Présence
Principe d'intensification	Présence	Présence	Présence
Dépendance au problème	Fortement lié	Adaptable aux différents problèmes	Applicable aux différents problèmes
Niveau d'abstraction	Bas	Haut	Haut
Espace de recherche	Sur l'espace d'état du problème	Sur les solutions du problème	Sur les méthodes de résolution présente dans sa base

7. Conclusion

Le développement des méthodes approchées est indispensable pour aborder des problèmes d'optimisation de grande taille. Les méthodes de recherche heuristique ont été les premières à montrer le chemin pour aborder la résolution de problèmes de façon approchée de l'optimum, de ce fait, il se peut qu'on puisse perdre dans la qualité de la solution, mais on gagne en temps de réponse. L'inconvénient de ces méthodes est qu'elles sont spécifiques à un problème, de ce fait, vouloir résoudre ces problèmes relève d'un niveau d'expertise très grand, ce qui a conduit à l'élaboration de méthodes d'un haut niveau d'abstraction, indépendantes du problème, et donc adaptables à ces dernier.

Plusieurs méthodes approchées ont été développées pour la résolution de diffèrent problème d'optimisation combinatoires. Ces méthodes possèdent des forces et des faiblesses ; ce qui a conduit à vouloir compenser la faiblesse des uns par la force des autres. Une méthode d'un plus haut rang, jugeant quelle méthode appliquer à tel instant, cette méthode est appelé hyper-heuristique.

Dans notre travail nous nous sommes intéressés à l'utilisation des hyper-heuristiques pour le problème du gagnant dans les enchères combinatoires.

Le chapitre suivant présentera les approches hyper-heuristiques proposées pour la résolution du PDG.

CHAPITRE III

Approches Proposées

1. INTRODUCTION

Dans ce chapitre nous présenterons trois hyper-heuristiques, à savoir *choice function* hyper-heuristique, *Random* hyper-heuristique et une méthode hybride combinant la fonction de choix (*choice function*) et la méthode de sélection aléatoire (*Random*) que nous appellerons *Stochastic Choice Function* Hyper-Heuristique. Ces méthodes sont des hyper-heuristiques de sélection à base d'heuristiques de bas niveau perturbatrice. Ces heuristiques de bas niveau sont au nombre de 5, utilisées dans la résolution du problème du gagnant dans les enchères combinatoires.

Nous pouvons structurer la présentation de nos approches en 2 parties distinctes, la première partie concerne les éléments liés au problème, et la deuxième partie traitera des hyper-heuristiques qui sont indépendant du problème traité.

2. PARTIE I

Dans cette partie nous parlerons des divers éléments liés au problème, avec la présentation des heuristiques de bas niveau développées pour ce problème et qui jouent un rôle important dans la résolution du PDG.

2.1. Représentation de la solution

Une solution est représentée par un vecteur de taille variable, où chaque élément de ce tableau représente une offre sélectionnée.

2.2. Solution initiale

Pour générer une solution initiale, nous avons utilisé la technique de l'encodage à clef aléatoire, dont le principe est décrit dans le chapitre 1.

2.3. Graphe de conflit

Afin de pouvoir générer des solutions admissibles, les algorithmes utilisés pour la résolution du problème des enchères combinatoires se basent sur le graphe de conflit qui donne la possibilité de savoir si une offre est en conflit avec une autre. Dans ce graphe les nœuds représentent les offres et les arcs sont les objets partagés par les offres, et par conséquent, sont en conflit.

2.4. Graphe d'offre non en conflit

Ce graphe a pour but d'extraire pour chaque offre un ensemble d'offres qui ne soient pas en conflit avec elle. Par contre les éléments de cet ensemble peuvent être en conflit entre eux.

2.5. Fonction objectif

Le problème du gagnant dans les enchères combinatoires consiste à trouver un ensemble d'offres qui maximise le profit du vendeur. Une condition sine qua non pour la validité de cet ensemble est du fait qu'aucun objet n'est présent dans plusieurs offres de cet ensemble. Pour pouvoir calculer le profit de ces ensembles on utilise la fonction suivante :

$$\sum_{i=1}^{L} Prix(Offre_i) \qquad avec\ L\ la\ taille\ de\ l'ensemble.$$

2.6. Heuristiques de bas niveau

Nous avons utilisé dans notre travail 5 heuristiques de bas niveau qui sont les seules à pouvoir manipuler les solutions du problème. Ces heuristiques de bas niveau travaillent avec différentes stratégies de génération de nouvelle solution, et se basent sur un graphe de conflit afin d'assurer la génération de solution admissible. Ces méthodes, après avoir terminé leurs exécutions, renvoient la qualité de la solution générer afin de permettre à l'hyper-heuristique de juger si elle l'accepte ou la rejette.

Nous présentons ci-dessous les différentes heuristiques de bas niveau utilisées dans notre approche.

2.6.1. Heuristique de bas niveau H1

Cette heuristique de bas niveau travaille avec une technique de mutation afin de perturber la solution courante.

Cette méthode commence par choisir aléatoirement une offre X de la solution courante, puis, à partir du graphe des offres non en conflit, elle choisit une offre qui n'est pas en conflit avec l'offre X, et la remplace. Il se peut que cette nouvelle offre ajoutée à la solution puisse introduire des conflits avec les offres déjà présentes dans cette solution, donc une vérification est nécessaire afin de supprimer ces conflits.

Une méthode de recherche locale est appliquée sur cette nouvelle solution afin de parcourir son voisinage dans le but de trouver mieux.

Algorithme de l'heuristique de bas niveau H1

Entrée : Solution A.

Sortie : Solution A', le profit de la solution A'.

Choisir une offre X dans A de manière aléatoire ;

A partir du graphe des offres non en conflit, prendre une offre Y aléatoirement qui ne soit pas en conflit avec X. ;

Enlever les offres en conflit avec Y. On obtient la solution A'' ;

Chercher le meilleur voisin de A'', appelé A' ;

F = Evaluation(A') ;

Retourner : A', F ;

2.6.2. Heuristique de bas niveau H2

Cette méthode utilise un mécanisme de croisement entre la meilleure solution trouvée pour le moment (notée A) et une solution extraite à partir du graphe des offres non en conflit (notée V) puis mutée en utilisant le même mécanisme de mutation que l'algorithme précédent.

Le résultat obtenu passera par une méthode de recherche locale, afin d'en trouver une meilleure solution dans son voisinage.

Algorithme de génération de la solution V

F_V=0 ;

V = Ø ;

Pour i=0 à m-1 // parcourir toute les offres du problème.

Faire

Construire une solution V', en se basant sur le graphe des offres non en conflit pour l'offre i, et vérifier que l'élément ajouté ne provoque pas de conflit, sinon on l'ignore.

$F_{V'}$ = Evaluation(V').

Si ($F_V < F_{V'}$) **alors**

V = V' ;

$F_V = F_{V'}$;

Fsi ;

i++ ;

Fait ;

Ci-dessous l'algorithme de l'heuristique 2 :

Algorithme : Heuristique de bas niveau H2

Entrée : Une solution V à partir du graphe des offres non en conflit, solution courante : A.

Sortie : Solution A', profit de A'.

A'= Ø ;

Choisir une offre X dans V de manière aléatoire ;

A partir du graphe des offres non en conflit, prendre une offre Y aléatoirement qui ne soit pas en conflit avec X. ;

Enlever les offres en conflit avec Y. On obtient la solution V' ;

Tantque (tous les éléments de A n'ont pas été explorés) **et** (tous les éléments de V' n'ont pas été explorés) **faire**

p= nombre aléatoire entre 0 et 1 ;

Si (p > 0.5) **alors**

Prendre un élément de A qui ne soit pas en conflit avec A'.

Sinon Prendre un élément de V' qui ne soit pas en conflit avec A'.

Fait ;

S'il reste des éléments non explorés dans A ou V', alors prendre les éléments qui ne soient pas en conflit avec les éléments de A'.

Appliquer la recherche locale sur A', et prendre le meilleur voisin.

F = Evaluation (A') ;

Retourner : A', F

2.6.3. Heuristique de bas niveau H3

Cette heuristique de bas niveau n'est autre que la métaheuristique SLS qui est, rappelons-le, une méthode de recherche locale stochastique qui, d'après un paramètre aléatoire, effectue une action sur la solution courante.

L'algorithme de cette métaheuristique est présenté dans le chapitre 1.

2.6.4. Heuristique de bas niveau H4

Cette méthode, comme pour l'heuristique 1, utilise la technique de mutation sur la solution courante afin d'en générer une nouvelle.

La différence entre cette méthode et la première, c'est que celle-ci possède un taux de mutation appelé TM. Si la probabilité p est inférieure à TM, alors l'algorithme choisit aléatoirement une offre X, puis procède à la mutation de la solution courante en choisissant aléatoirement une offre Y lui appartenant qui sera remplacé par X, puis enlève toutes les offres en conflit avec X. Enfin, la nouvelle solution passe par une méthode de recherche locale afin de voir s'il y a moyen de trouver une bonne solution dans son voisinage.

Algorithme : Heuristique de bas niveau H4

Entrée : Solution A.

Sortie : Solution A', le profit de la solution A'.

Soit TM le taux de mutation de la méthode dont la valeur est déterminée empiriquement ;

r= nombre aléatoire entre 0 et 1 ;

Si (r < TM) **alors**

Choisir une offre X aléatoirement ;

Choisir une offre Y dans A de manière aléatoire ;

Enlever les offres en conflit avec X. ;

Remplacer Y par X, On obtient la solution A'' ;

Chercher le meilleur voisin de A'', appelé A' ;

F = Evaluation(A') ;

Fsi ;

Retourner : A', F ;

2.6.5. Heuristique de bas niveau H5

Tout comme H2, H5 utilise un mécanisme de croisement pour générer une nouvelle solution, mais cette fois ci elle croise la meilleure solution trouvée avec une solution générée aléatoirement en utilisant l'encodage de clef aléatoire (présenté dans le chapitre 1)

Algorithme : Heuristique de bas niveau H5

Entrée : La solution courante : A.

Sortie : Solution A', profit de A'.

A'= Ø ;

Générer une solution aléatoire V' à l'aide du *Random Key Encoding* (RK).

Tantque (tous les éléments de A n'ont pas été explorés) **et** (tous les éléments de V' n'ont pas été explorés) **faire**

p= nombre aléatoire entre 0 et 1 ;

Si (p > 0.5) **alors**

Prendre un élément de A qui ne soit pas en conflit avec A'.

Sinon Prendre un élément de V' qui ne soit pas en conflit avec A'.

Fait ;

S'il reste des éléments non explorés dans A ou V', alors prendre les éléments qui ne soient pas en conflit avec les éléments de A'.

Appliquer la recherche locale sur A', et prendre le meilleur voisin.

F = Evaluation (A') ;

Retourner : A', F

3. PARTIE II

Dans cette partie nous parlerons des approches indépendantes du problème.

Dans le but de résoudre le problème du PDG, nous avons proposé trois approches hyper-heuristiques : CF-HH, R-HH et SCF-HH.

3.1. L'hyper-heuristique CF-HH

CF-HH est l'abréviation de « *Choice function-Only Improvement Hyper-heuristic* ». Comme son nom l'indique, elle est composée d'une méthode de sélection d'heuristique appelée *fonction de choix* ou *choice function*, ainsi que d'une méthode d'acceptation de solution qui ne valide que les solutions qui améliorent la solution courante (Only improvement acceptance).

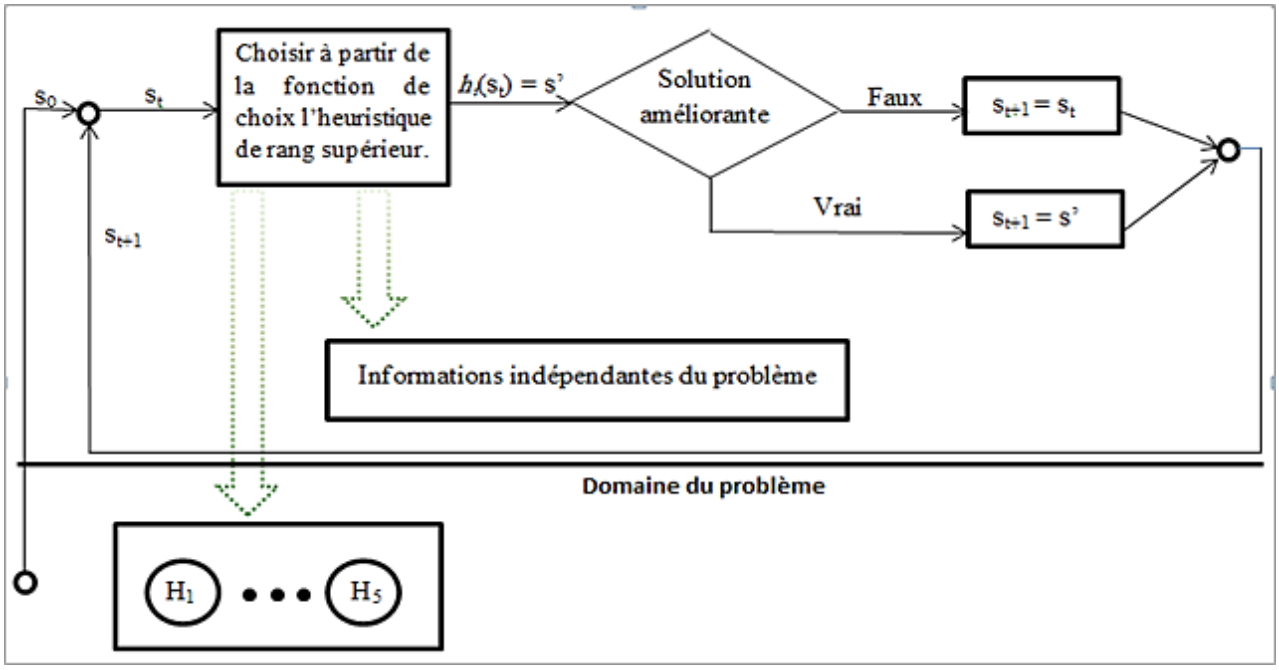

Figure 14. Schema global de l'hyper-heuristique CF-HH

Choice function **[17]** est une technique basée score, qui attribue à chaque heuristique de bas niveau un poids. En effet, cette technique permet de juger sur l'efficacité d'une heuristique de bas niveau afin de décider laquelle sera exécutée pour la prochaine exécution. Cette technique se base sur des données qui sont indépendantes du problème. Ces données sont : le temps CPU lors de l'exécution, la qualité de la solution, ainsi que le temps écoulé depuis que l'heuristique de bas niveau eut été appelée pour la dernière fois.

Afin de pouvoir classer ces heuristiques de bas niveau, choice function est définie comme suit :

$$\forall i, g_1(h_i) = \sum_n \alpha^{n-1} \frac{I_n(h_i)}{T_n(h_i)} \quad (1)$$

$$\forall i, g_2(h_{ID}, h_i) = \sum_n \beta^{n-1} \frac{I_n(h_i, h_{ID})}{T_n(h_i, h_{ID})} \quad (2)$$

$$\forall i, g_3(h_i) = elapsedTime(h_i) \quad (3)$$

$$\forall i, score(h_i) = \alpha g_1(h_i) + \beta g_2(h_{ID}, h_i) + \delta g_3(h_i) \quad (4)$$

Avec $\alpha, \beta \in]0,1]$ $\delta \in \mathbb{R}$.

$I_n(h_i)$ (respectivement, $I_n(h_i,h_{ID})$) représente le changement dans la fonction d'évaluation de hi après sa néme exécution (respectivement, juste après l'exécution de

h_{ID}), et T_n (h_i) (respectivement, T_n (h_i,h_{ID})) représente le temps d'exécution de hi après son néme appel (respectivement, juste après l'exécution de h_{ID}).

L'équation (1) permet d'évaluer la performance individuelle de l'heuristique de bas niveau, en prenant en compte ses n dernières fois où elle a été appelée. Cette fonction n'est pas suffisante pour juger l'efficacité d'une heuristique de bas niveau dans un état donné, c'est pour cette raison que l'équation (2) mesure la synergie entre les heuristiques de bas niveau (h_{ID} représente la dernière heuristique de bas niveau exécuté). Si une heuristique de bas niveau n'a pas pu être appelée pendant longtemps, sachant qu'à un moment donné son exécution pouvait produire de bons résultats- ce qui représente un paramètre important- et c'est ce que fait l'équation (3), qui calcule le temps écoulé depuis sa dernière exécution. L'équation (4) collecte les résultats des 3 fonctions présentées précédemment et calcule pour chaque heuristique de bas niveau son score.

Cette stratégie de sélection d'heuristique de bas niveau est très appréciée au vu des bons résultats fournis dans divers problèmes **[12][47]**.

Dans notre travail, choice function a été associée à une méthode d'acceptation de solution améliorante (Only Improvement). Contrairement à certaine méthode pouvant accepter des solutions non améliorantes (avec un certain degré d'acceptation de la détérioration de la solution courante, exemple : Recuit Simulé, Monté Carlo,…), Only Improvement est une méthode déterministe, possédant une démarche simple, sans paramètre (contrairement au Recuit Simulé) et facile à implémenter.

L'algorithme de l'hyper-heuristique choice function-Only Improvement est présenté comme suit :

Algorithme CF-HH

Entrée : Instance du problème, ensemble d'heuristique de bas niveau : HBN, α, β, δ, max_iter.

Sortie : une solution du problème (dans notre cas une allocation)

//Initialisation.

Iter=0 ;

Generation du graphe de conflit ; Generation du graphe des offres qui ne sont pas en conflit ;

Generer une solution complète A aléatoirement à l'aide de RK, et retourne son evaluation F_{best}.

Tantque (iter < max_iter)

Faire

//Methode de selection d'heuristique.

A = A_{best} ;

i= choice function (HBN) ; // calcule le score de chaque heuristique de bas niveau, et renvoie celui avec le score le plus grand.

appliquer l'heuristique i à la solution A. i retourne la nouvelle solution A' avec son evaluation F.

enregistrer les performances de "i" (individuel et avec le groupe).

calcule de la fonction (3) pour le reste des heuristiques de bas niveau.

// Methode d'acceptation.

Si (F > F_{best}) **Alors**

F_{best} = F ;

A_{best}= A ;

Fsi ;

iter++;

Fait ;

Retourner : A_{best} ;

3.2. L'hyper-heuristique R-HH

Contrairement à l'hyper-heuristique présentée précédemment, R-HH ou *Random-Only Improvement Hyper-heuristic* n'est pas une méthode informée, c'est-à-dire, elle n'utilise aucune information lui permettant de choisir intelligemment une heuristique de bas niveau.

Cette hyper-heuristique est une hyper-heuristique composée de deux modules : une méthode de sélection d'heuristique de bas niveau d'une manière non-informé, et d'une méthode d'acceptation de solution améliorante seulement.

La méthode de sélection procède par la sélection d'heuristique de manière aléatoire sans tenir en compte l'historique de son exécution, ni de combien de temps elle n'a pas été appelé.

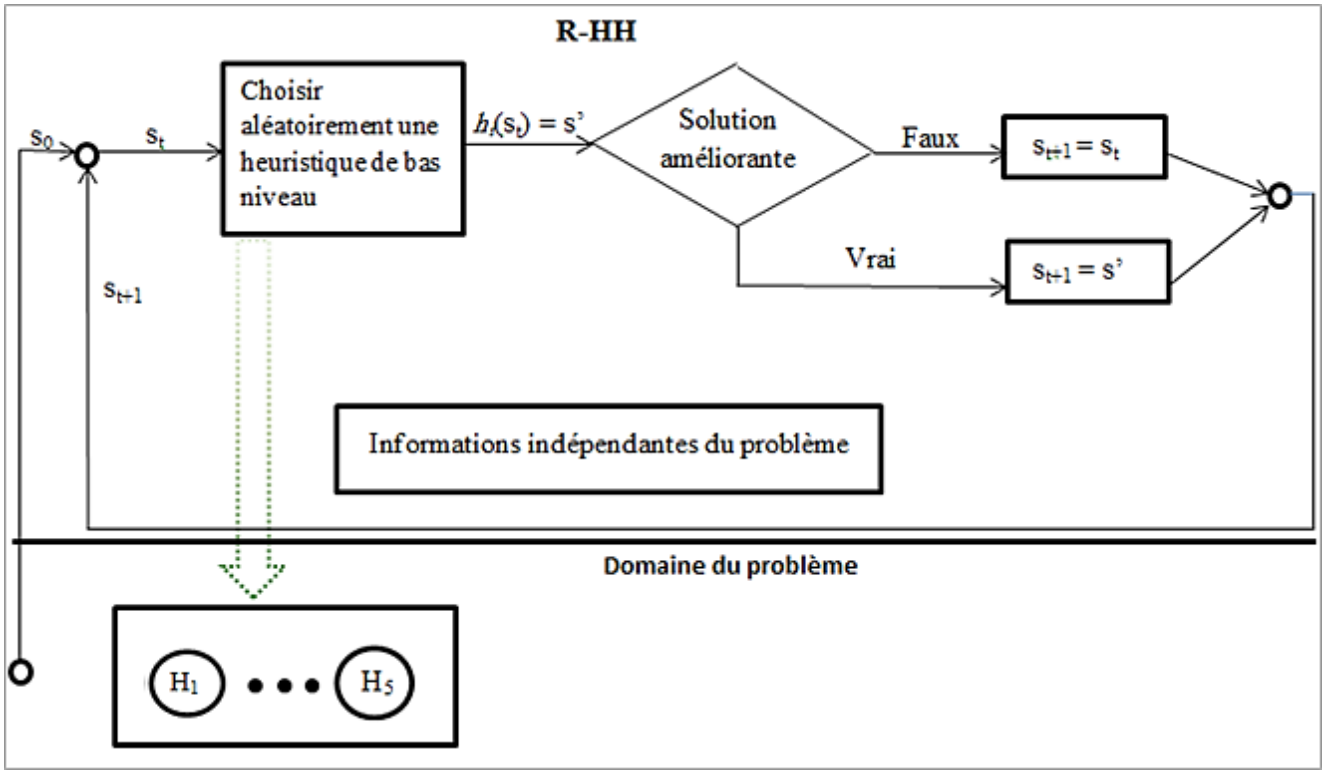

Figure 15. Schema global de l'hyper-heuristique R-HH

La méthode d'acceptation utilisée est la même que celle utilisée dans CF-HH. L'algorithme du Random-Only Improvement Hyper-heuristique est présenté comme suit :

Algorithme : R-HH

Entrée : Instance du problème, ensemble d'heuristiques de bas niveau, max_iter ;

Sortie : Solution du problème ;

// initialisation

Iter=0 ;

Generation du graphe de conflit ;

Generation du graphe des offres qui ne sont pas en conflit ;

Generer une solution complete A aléatoirement à l'aide de RK, et retourner son

evaluation F_{best}.

Tantque (iter < max_iter)

Faire

//Méthode de sélection d'heuristique.

$A=A_{best}$;

Choisir une heuristique de bas niveau "i" de façon aléatoire.

Appliquer l'heuristique i à la solution A. i retourne la nouvelle solution A' avec son évaluation F.

// Méthode d'acceptation.

Si ($F > F_{best}$) **Alors**

$F_{best} = F$;

$A_{best}= A$;

Fsi ;

iter++;

Fait ;

Retourner : A_{best}.

3.3. L'hyper-heuristique SCF–HH

Dans cette Hyper-heuristique deux techniques de sélection d'heuristique de bas niveau ont été utilisées. Il s'agit de la technique du choice function et de la méthode de sélection aléatoire ; ainsi nous avons combiné une méthode informée avec une autre "*aveugle*".

Un paramètre est utilisé pour décider à quelle méthode faire appel. Ce paramètre est appelé *m_choix*. Si la probabilité p est inférieure à *m_choix*, alors la méthode de sélection aléatoire est appliquée. Sinon, la méthode de sélection à base de score est appliquée afin de choisir l'heuristique de bas niveau qui possède le score le plus grand.

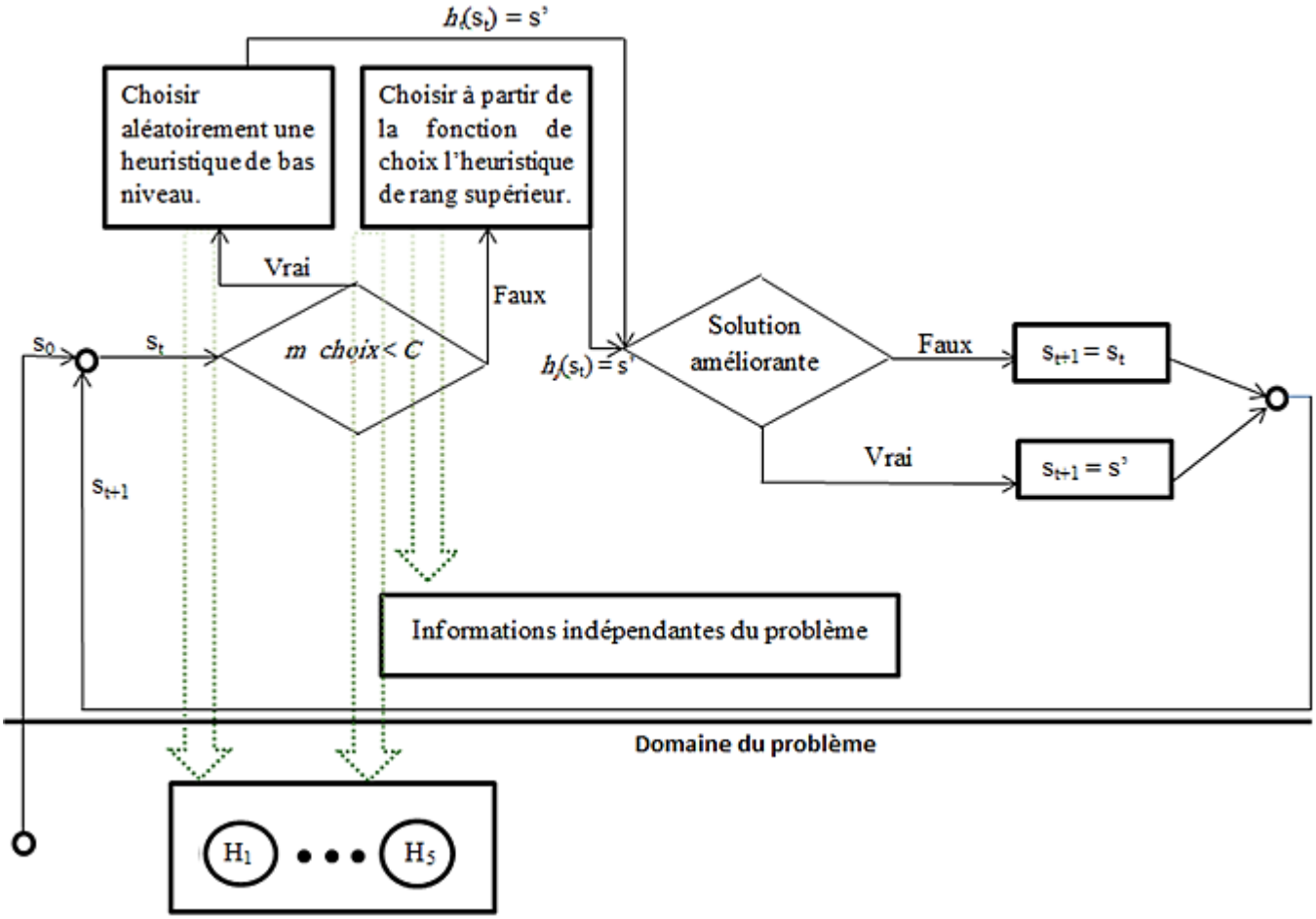

Figure 16. Schema global de l'hyper-heuristique SCF-HH

Il est à préciser qu'après l'exécution d'une heuristique de bas niveau sélectionnée grâce à la méthode aléatoire, ses performances sont quand même sauvegardées afin de permettre à la fonction de choix de continuer à collecter les informations concernant les performances de ces méthodes de résolution.

Pour ce qui est de la méthode d'acceptation de solution, elle est la même que celle utilisée dans CF-HH et R-HH.

Algorithme : SCF-HH

Entrée : Instance du problème, ensemble d'heuristiques de bas niveau : HBN, α, β, δ, *max_iter*, *m_choix*.

Sortie : une solution du problème (dans notre cas une allocation)

//Initialisation.

Iter=0 ;

Génération du graphe de conflit ; Génération du graphe des offres qui ne sont pas en conflit ;

Générer une solution complète A aléatoirement à l'aide de RK, et retourne son évaluation F_{best}.

Tantque (iter < max_iter)

Faire

//Methode de selection d'heuristique.

$A = A_{best}$;

p = nombre aléatoire entre 0 et 1 ;

Si (p < *m_choix*) **alors**

choisir une heuristique de bas niveau "i" aléatoirement ;

Sinon i= choice function (HBN) ;

// calcule le score de chaque heuristique de bas niveau, et renvoie celui avec le score le plus grand.

Fsi ;

Appliquer l'heuristique i à la solution A. i retourne la nouvelle solution A' avec son évaluation F.

Enregistrer les performances de i (individuel et avec le groupe), calcule de la fonction (3) pour le reste des heuristiques de bas niveau.

// Méthode d'acceptation.

Si ($F > F_{best}$) **Alors**

$F_{best} = F$;

$A_{best} = A$;

Fsi ;

```
        iter++;
    Fait ;
Retourner : A_best.
```

4. Conclusion

Une nouvelle page s'ouvre dans la résolution du problème du PDG, avec l'exploitation de plusieurs méthodes en même temps, au lieu de l'utilisation d'une méthode unique. Cette exploitation de méthode offre une diversité dans l'exploration de l'espace de recherche, rendant ainsi possible l'exploration de zones très intéressantes et difficiles d'accès si on utilisait une seule méthode.

Cette façon d'aborder la résolution du problème du PDG sera comparée avec une métaheuristique, à savoir la recherche locale stochastique SLS dont les résultats expérimentaux sont présentés dans le chapitre suivant.

CHAPITRE IV

Résultats expérimentaux

1. INTRODUCTION

Dans ce chapitre, nous présenterons les différents résultats obtenus pour chaque hyper-heuristique développée. La meilleure méthode et la mauvaise seront comparées à la méthode de recherche locale stochastique présentée dans le chapitre 1.

2. RESULTATS EXPERIMENTAUX

2.1. Caractéristiques de la machine

Les tests ont été faits sur une machine avec un système d'exploitation Windows 7, qui possède un processeur itel CORE i7 avec 2.40 GHz pour chaque cœur, 8 GO de mémoire vive.

Les algorithmes développés pour notre projet sont implémentés en langage C à l'aide de l'IDE code::blocks version 10.05.

2.2. Benchmarks

Une batterie de tests de 500 instances ont été utilisés pour évaluer les performances des différents algorithmes et ainsi pouvoir les comparer entre eux. Ces instances sont fourni par Lau et Goh **[31]**, qui divisent ces 500 instances en 5 groupes de 100, ces groupes se distinguent entre eux par le nombre d'offres présentes, noté m, et par le nombre d'objets à vendre, noté n. Ces 5 groupes sont présentés comme suit :

- **REL-1000-500 :** ce groupe contient un ensemble de 100 instances de m=500 objets et de n=1000 offres. Dans ce groupe les instances sont identifiées de in101 jusqu'à in200.
- **REL-1000-1000 :** ce groupe contient un ensemble de 100 instances de m=1000 objets et de n=1000 offres. Dans ce groupe les instances sont identifiées de in201 jusqu'à in300.
- **REL-500-1000 :** ce groupe contient un ensemble de 100 instances de m=1000 objets et de n=500 offres. Dans ce groupe les instances sont identifiées de in401 jusqu'à in500.
- **REL-1000-1500 :** ce groupe contient un ensemble de 100 instances de m=1000 objets et de n=1500 offres. Dans ce groupe les instances sont identifiées de in501 jusqu'à in600.

- **REL-1500-1500 :** ce groupe contient un ensemble de 100 instances de m=1500 objets et de n=1500 offres. Dans ce groupe les instances sont identifiées de in601 jusqu'à in700.

2.3. Valeurs des paramètres

Plusieurs algorithmes sont concernés par le paramétrage, les valeurs de ces paramètres sont choisies de façon expérimentale où il a fallu tester plusieurs valeurs afin de trouver celles qui conviennent le mieux.

Comme nous l'avons cité dans le chapitre 2, un bon paramétrage joue un rôle très important dans l'efficacité d'un algorithme qui lui permet de garder un certain équilibre de la balance de l'exploration et l'exploitation de l'espace de recherche.

Tout d'abord, les paramètres utilisés dans les différentes heuristiques de bas niveau sont présentés comme suit :

- TM = 0.3 : qui représente le taux de mutation utilisé dans l'heuristique H2.
- wp = 0.3 : paramètre utilisé dans l'heuristique H3.
- R1 = 0.5 : paramètre utilisé dans les heuristiques de bas niveau H5 et H1 pour l'opération de croisement entre deux solutions.

Les paramètres utilisés par les hyper-heuristiques (sauf pour l'hyper-heuristique stochastique R-HH) sont présentés comme suit :

- α =0.9, β =0.1, δ =1.5 : ces trois paramètres sont utilisés par SCF-HH et CF-HH pour la sélection d'heuristique de bas niveau d'après leurs performances antérieures.
- m_choix = 0.0335 : qui est le paramètre utilisé par l'hyper-heuristique SCF-HH afin de choisir quelle méthode de sélection d'heuristique de bas niveau appliquer. La qualité de la solution croît lorsque le paramètre m_choix se rapproche de la valeur 0.0335, mais ce qui n'est pas le cas du temps d'exécution qui augmente légèrement et de manière proportionnelle à m_choix. Puisque le temps d'exécution n'augmente que légèrement, ceci nous a conduit à prendre cette valeur afin de pouvoir améliorer la solution des différentes instances du problème.
- Pour le nombre d'itération *max_iter*, il diffère pour chaque hyper-heuristique. Ainsi nous avons :
  - *max_iter* = 1 000 000 itérations pour CF-HH.
  - *max_iter* = 500 000 itérations pour SCF-HH.

- *max_iter* = 50 000 itérations pour R-HH.

3. RÉSULTATS

Dans cette partie nous présenterons les résultats des trois hyper-heuristiques appliquées au problème du PDG à partir des 5 groupes d'instances présentées précédemment. Ces résultats sont présentés sous forme de tableaux où chacun d'eux correspond à un groupe d'instance. Ces tableaux sont composés de trois colonnes qui représentent les trois hyper-heuristiques et chaque ligne correspond à une instance du groupe.

Les résultats donnés par l'hyper-heuristique R-HH sont les résultats donnés après une seule série d'exécutions pour chaque instance, puisque la qualité de la solution ne change pas pour d'autres exécutions. Par contre, pour l'hyper-heuristique SCF-HH et CF-HH c'est la moyenne des 10 exécutions qui sont présentées pour chaque instance.

Ci-après, les tableaux de 2 à 6 présentent les résultats des 13 premières instances ainsi que des 13 dernières de chaque groupe pour les différents algorithmes hyper-heuristiques du problème du PDG, où pour chaque colonne du tableau, deux sous colonnes correspondent au temps d'exécution et le profit généré par la solution. Les meilleurs résultats sont en gras.

Tableau 2. Comparaison entre CF-HH, SCF-HH et R-HH sur quelques instances du groupe REL-500-1000

Instances	CF-HH		SCF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
in101	28,92	66 170,61719	26,5029	68 315,63750	**18,18**	**68 619,06250**
in102	32,21	67 820,63438	22,6434	**68 428,89453**	**15,54**	64 800,01563
in103	36,38	66 488,51016	22,6433	68 236,46250	**15,5**	**68 983,25000**
in104	31,44	66 290,84141	23,9882	**68 596,57188**	**14,05**	61 949,51563
in105	29,99	67 126,14063	25,2727	**70 111,21172**	**17,33**	64 438,91406
in106	29,55	65 452,02930	22,6527	**66 554,30469**	**15,83**	65 117,76953
in107	28,99	68 524,32344	22,4421	**68 587,17188**	**15,22**	59 622,38281
in108	29,91	71 529,20313	24,8401	**74 506,78750**	**16,89**	64 000,28906
in109	34,25	65 680,75664	22,5529	**66 203,72656**	**16,24**	59 982,78516
in110	31,76	63 945,89453	24,4317	**64 867,79063**	**16,78**	63 566,44922
in111	39,02	71 274,03281	25,2159	**72 969,16406**	**16,52**	72 969,16406
in112	29,30	65 699,98125	24,0179	**67 163,64844**	**15,83**	65 590,26563
in113	30,18	**68 618,92110**	23,815	68 115,35626	**16,33**	64 122,30469
in188	29,06	66 149,44141	23,0943	**67 662,82070**	**17,209**	61 769,67188
in189	29,61	67 330,77344	27,2877	**68 337,96875**	**17,009**	63 346,44922
in190	36,16	67 490,41485	22,453	**68 022,94141**	**16,667**	61 861,35938
in191	30,93	69 552,65313	23,8697	**71 236,99375**	**16,617**	61 434,90234
in192	28,96	**66 808,53906**	23,4326	66 016,80039	**14,892**	57 668,03125
in193	29,59	67 076,20234	21,9646	**67 885,93750**	**15,682**	60 857,00781
in194	29,76	68 954,93750	23,9865	**69 687,35703**	**16,889**	62 808,57031
in195	29,47	**69 066,85938**	23,2971	66 015,40156	**15,989**	61 907,14844
in196	34,89	68 655,61719	23,3816	**68 707,62422**	**16,47**	60 672,15625
in197	32,11	62 824,19531	22,383	**63 307,77031**	**15,306**	56 949,31250
in198	30,79	66 777,95626	25,3048	**69 717,45625**	**16,523**	66 144,22656
in199	28,96	**69 866,03125**	23,0117	69 711,90859	**16,277**	62 216,54688
in200	34,48	**72 291,80313**	22,7231	71 353,74297	**15,318**	66 243,24219

D'après les résultats trouvés dans le tableau 2, la qualité de la solution trouvée SCF-HH est plus performante que CF-HH et R-HH, et dans un temps assez raisonnable pour le groupe **REL-500-1000**. R-HH trouve rapidement des solutions mais ne sont pas performantes dans la majorité des instances.

Tableau 3. Comparaison entre CF-HH, SCF-HH et R-HH sur quelques instances du groupe REL-1000-1000

Instances	CF-HH		SCF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
in201	27,89	77 056,48438	16,14	**81 387,33047**	**14,34**	70 388,03125
in202	20,45	87 003,51563	17,63	**90 408,93047**	**13,85**	76 900,35938
in203	20,55	81 696,49219	16,44	**86 239,21094**	**14,09**	77 169,17188
in204	20,17	81 969,04688	16,64	**82 575,65859**	**14,21**	79 241,41406
in205	19,18	**82 571,21016**	15,84	81 622,47656	**14,01**	72 614,98438
in206	19,92	82 935,10469	16,17	**86 436,02579**	**14,25**	76 224,60156
in207	24,31	**91 033,51563**	17,23	**91 033,51563**	**14,13**	89 048,46875
in208	25,77	**88 477,66797**	16,95	85 173,28906	**14,20**	85 173,28906
in209	19,61	**86 462,06562**	16,09	86 117,06719	**14,35**	78 044,66406
in210	19,96	**85 079,98438**	15,85	**85 079,98438**	**13,99**	82 949,44531
in211	19,28	80 240,12969	16,36	**81 359,82344**	**14,52**	80 752,77344
in212	21,81	**81 459,96094**	16,39	81 229,52734	**14,66**	78 715,03906
in213	20,44	**82 271,00000**	16,51	82 253,71797	**15,58**	73 899,15625
in288	18,81	**82 909,68750**	15,84	**82 909,68750**	**14,45**	78 303,71875
in289	23,43	**81 619,98360**	16,71	80 930,92657	**15,49**	76 246,18750
in290	19,77	**83 985,00000**	16,70	83 263,24766	**13,45**	77 931,26563
in291	37,82	88 741,33828	17,51	**89 388,06797**	**14,74**	80 111,68750
in292	21,02	89 973,68203	16,92	**89 974,89453**	**14,87**	84 571,60938
in293	24,71	**81 964,96875**	16,88	81 301,78282	**14,78**	76 516,00781
in294	37,14	85 637,27422	16,47	**88 174,37578**	**14,34**	77 175,76563
in295	26,44	83 086,79688	15,78	**85 770,60938**	**13,66**	75 693,48438
in296	19,40	**85 220,38281**	15,78	84 254,82656	**14,55**	79 936,94531
in297	21,25	**81 796,47110**	16,57	81 542,87891	**14,13**	76 160,31250
in298	21,83	83 216,29766	16,24	**83 847,04297**	**12,98**	81 654,10156
in299	27,98	83 422,37579	17,21	**89 143,13516**	**14,16**	76 252,15625
in300	23,44	**88 373,32031**	16,89	**88 373,32031**	**14,74**	79 439,40625

Le tableau3 donne les résultats trouvés des 3 hyper-heuristiques pour le groupe **REL-1000-1000**, et ils nous montrent l'efficacité de SCF-HH et CF-HH où ces deux méthodes s'imposent sur R-HH, avec un avantage dans le temps d'exécution pour SCF-HH.

Tableau 4. Comparaison entre CF-HH, SCF-HH et R-HH sur quelques instances du groupe REL-1000-500

Instances	CF-HH		SCF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
in401	9,52	72 948,07031	**6,70**	**75 588,99610**	14,34	68 009,35938
in402	9,28	**72 820,03125**	**6,75**	71 693,71953	13,85	71 454,78906
in403	9,24	**74 843,96094**	**6,73**	**74 843,96094**	14,09	67 873,03125
in404	9,64	76 567,12539	**6,86**	**78 761,68750**	14,21	69 322,72656
in405	9,52	71 636,86641	**7,20**	**74 806,79688**	14,01	74 138,28125
in406	8,54	67 862,87734	**6,47**	**71 791,03125**	14,25	71 009,36719
in407	10,59	73 699,60156	**7,27**	**73 935,28125**	14,13	69 484,14844
in408	9,98	77 018,83594	**6,79**	**77 018,83594**	14,20	69 729,71094
in409	10,64	67 791,67031	**6,28**	**69 102,72266**	14,35	65 251,77344
in410	10,20	71 791,57813	**7,02**	**71 791,57813**	13,99	71 791,57813
in411	8,91	69 335,00508	**6,84**	**71 272,47656**	14,52	71 272,47656
in412	10,95	73 351,24297	**6,17**	**75 292,63281**	14,66	75 292,63281
in413	9,97	**73 494,55469**	**6,95**	72 561,90625	15,58	72 561,90625
in488	10,16	77 928,14844	**6,34**	**77 928,14844**	14,45	70 299,06250
in489	9,17	71 306,45938	**6,56**	**73 740,02969**	15,49	67 364,98438
in490	8,62	70 108,95860	**6,35**	**71 423,73438**	13,45	63 984,82031
in491	9,40	72 163,23633	**6,97**	**76 284,35625**	14,74	71 905,09375
in492	9,76	**72 448,45157**	**7,63**	72 318,21875	14,87	70 484,99219
in493	10,10	74 545,88515	**6,63**	**76 953,24766**	14,78	76 170,53125
in494	9,87	74 225,70625	**6,79**	**74 398,20313**	14,34	66 717,17188
in495	8,96	71 911,85938	**6,49**	**72 919,67969**	13,66	72 919,67969
in496	10,27	73 501,33360	**7,43**	**73 533,67344**	14,55	73 376,74219
in497	9,23	74 480,69531	**7,28**	**76 076,26563**	14,13	72 196,35156
in498	9,11	**80 569,27891**	**6,32**	74 213,25000	12,98	67 302,00781
in499	9,59	71 246,87500	**6,95**	**72 113,62266**	14,16	63 359,17188
in500	9,82	70 717,49610	**6,59**	**73 320,69063**	14,74	66 948,61719

Les résultats du groupe **REL-1000-500** sont illustrés dans le tableau4, où SCF-HH est le plus efficace dans presque toutes les instances du groupe avec une exécution plus rapide que les deux autres hyper-heuristiques.

Tableau 5. Comparaison entre CF-HH, SCF-HH et R-HH sur quelques instances du groupe REL-1000-1500

Instances	CF-HH		SCF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
in501	**22,79**	79 453,92969	23,02	**79 864,55937**	29,08	77 738,97656
in502	**23,00**	80 340,76563	23,02	**80 340,76563**	28,87	76 348,53906
in503	25,03	79 514,52813	**23,57**	**83 277,71094**	28,13	79 640,48438
in504	25,43	81 903,02344	**22,90**	**81 903,02344**	27,54	77 668,24219
in505	**22,96**	77 977,65625	22,97	**77 977,65625**	27,33	77 977,65625
in506	**20,77**	76 775,33281	21,43	**77 523,67656**	28,05	77 254,13281
in507	**24,34**	84 174,39063	25,07	**84 174,39063**	29,36	84 174,39063
in508	**22,54**	80 446,06250	23,36	**82 149,27344**	28,43	82 149,27344
in509	25,43	82 672,25781	**23,49**	**82 672,25781**	30,13	76 472,35156
in510	22,34	78 601,80625	**21,62**	**80 102,35000**	27,19	74 997,00000
in511	**20,29**	79 754,37500	22,10	**78 793,83594**	26,90	78 793,83594
in512	**22,83**	82 342,70313	22,91	**82 445,79687**	28,89	78 998,21094
in513	**22,23**	77 142,01094	23,53	**84 079,56250**	29,47	84 079,56250
in588	24,38	81 571,06758	**23,73**	84 770,49219	28,56	**85 201,01563**
in589	26,32	77 504,28125	**25,04**	**80 834,24219**	29,03	80 834,24219
in590	23,45	79 032,71797	**23,11**	**79 170,10156**	28,02	79 170,10156
in591	**19,62**	80 227,53125	21,56	**80 227,53125**	26,92	77 586,61719
in592	25,65	78 381,93750	**23,05**	**80 131,55547**	27,77	76 314,15625
in593	**23,31**	81 262,28125	24,25	**86 411,11719**	31,16	81 859,21875
in594	**22,28**	86 112,90625	22,42	**86 112,90625**	28,86	80 210,50000
in595	23,51	77 582,07813	**23,32**	**85 061,31250**	28,64	77 402,39063
in596	**19,23**	77 878,38281	21,21	**78 967,11563**	27,80	73 977,00781
in597	23,07	79 320,49219	**22,54**	**80 443,87500**	27,32	77 642,75781
in598	28,46	79 201,96875	**25,05**	**80 272,86328**	26,66	75 883,96875
in599	**21,78**	77 527,33594	22,04	**77 527,33594**	27,90	76 078,31250
in600	**20,80**	83 560,88750	23,73	**91 538,68750**	29,54	91 538,68750

Le tableau5 nous donne les résultats trouvés par SCF-HH, CF-HH et R-HH sur quelques instances du groupe **REL-1000-1500**. Nous constatons que SCF-HH donne dans la majorité des instances de bien meilleurs résultats que CF-HH et R-HH, et parfois les trois hyper-heuristiques trouvent le même résultat pour certaines instances du groupe.

Tableau 6. Comparaison entre CF-HH, SCF-HH et R-HH sur quelques instances du groupe REL-1500-1500

Instances	CF-HH		SCF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
in601	24,80	100 258,29375	**24,52**	**105 402,77266**	27,82	97 068,27344
in602	25,09	**98 164,23438**	**23,51**	95 328,21875	27,16	95 328,21875
in603	25,21	94 126,96094	**23,49**	**95 039,86719**	28,31	89 086,69531
in604	24,45	103 568,86719	**23,35**	**103 568,86719**	28,53	103 289,63281
in605	28,01	**102 404,76563**	**24,73**	101 470,54688	27,11	96 930,74219
in606	24,54	**105 218,21094**	**23,68**	104 346,07031	27,61	96 238,88281
in607	25,19	105 869,44531	**23,93**	**105 869,44531**	27,85	102 119,13281
in608	**23,73**	94 979,79063	23,83	**101 824,13281**	27,92	94 949,40625
in609	26,21	98 703,93281	**23,85**	**99 023,57031**	27,98	99 023,57031
in610	**26,78**	102 468,60156	27,08	**111 835,92188**	30,18	101 089,13281
in611	24,64	**106 352,51563**	**23,48**	98 826,87032	26,89	90 639,53906
in612	26,05	104 880,32656	**25,29**	**106 056,07813**	28,18	94 273,08594
in613	25,81	**95 529,18359**	**24,00**	93 371,29063	28,44	90 353,45313
in688	25,98	**104 657,50859**	**24,34**	103 959,20938	27,07	99 796,36719
in689	27,41	101 765,07031	**24,78**	**101 767,98437**	28,19	92 270,11719
in690	24,48	**103 340,47656**	**23,89**	103 094,79531	27,87	96 793,16406
in691	**23,43**	102 167,67969	23,61	**102 167,67969**	28,11	102 167,67969
in692	27,01	**99 712,39765**	**24,50**	98 865,29688	28,32	98 865,29688
in693	25,44	**100 162,28281**	**23,89**	99 616,92969	28,19	98 747,82813
in694	24,53	108 114,12500	**23,24**	**108 114,12500**	27,84	96 316,60938
in695	25,90	99 479,48515	**25,55**	**102 094,31406**	29,45	101 671,88281
in696	24,15	**98 035,48438**	**23,38**	97 351,56251	27,76	95 002,00781
in697	23,44	97 738,42188	**22,74**	**97 995,98438**	27,42	94 195,17188
in698	26,55	99 602,89844	**24,29**	**101 554,40313**	28,05	92 468,02344
in699	27,11	103 762,70313	**24,46**	**103 762,70313**	26,78	92 010,72656
in700	25,60	**101 510,20313**	**24,40**	97 675,50781	27,67	90 723,92969

Les résultats donnés par les 3 hyper-heuristiques sur certaines instances du groupe à 1500 objets et 1500 offres sont présentés dans le tableau 6. Ces résultats nous montrent que SCF-HH et CF-HH s'imposent sur R-HH. SCF-HH et CF-HH donnent parfois les mêmes résultats dans certaines instances du groupe, et dans d'autre cas un léger avantage de l'un par rapport à l'autre.

Tableau 7. Comparaison entre CF-HH, SCF-HH et R-HH sur la moyenne des 5 groupes d'instances du problème

Instances	SCF-HH		CF-HH		R-HH	
	Temps	Solution	Temps	Solution	Temps	Solution
REL-500-1000	23,82	**67 826,4412**	32,06	66 733,65985	**16,22**	62 563,9407
REL-1000-500	6,79	**73 859,7833**	9,69	73 137,22342	**4,02**	69 562,4518
REL-1000-1000	16,50	**85 065,3156**	23,19	84 147,87323	**14,40**	78 728,6097
REL-1000-1500	23,06	**81 843,9000**	**22,88**	80 414,73668	28,40	78 961,0330
REL-1500-1500	**24,12**	**101 804,8757**	25,46	101 071,41100	28,16	96 939,0750

Le tableau 7 nous résume les performances des algorithmes en nous donnant la moyenne du temps d'exécution ainsi que la qualité des solutions pour chaque groupe, ainsi nous remarquons que l'hyper-heuristique SCF-HH donne de meilleurs résultats en terme de qualité de la solution que les deux autres hyper-heuristiques. Nous pouvons analyser cela par le faite que SCF-HH comparé à CF-HH ajoute une perturbation dans le choix de la prochaine heuristique de bas niveau à exécuter, ceci s'est avéré efficace, puisque dans certaines situations un algorithme peut être plus efficace que les autres, offrant une meilleure exploration de l'espace de recherche. Si cet algorithme ne possède pas le meilleur score, il ne pourra pas être exécuté; c'est alors que cette perturbation intervient pour lui permettre de s'exécuter et ainsi pouvoir améliorer la solution courante et en même temps, améliorer son rang. R-HH n'offre pas vraiment de bons résultats en moyenne, ce qui nous permet de déduire que les méthodes de sélection intelligente sont plus efficaces que celles possédant un comportement aléatoire.

S'agissant du temps d'exécution, R-HH offre une exécution plus rapide que SCF-HH et CF-HH pour les groupes **REL-500-1000**, **REL-1000-500** et **REL-1000-1000**. CF-HH s'impose en moyenne sur le temps d'exécution pour le groupe **REL-1000-1500**, et la meilleure méthode, à savoir SCF-HH, s'exécute plus rapidement pour le groupe de grande taille avec 1500 offres et 1500 objets.

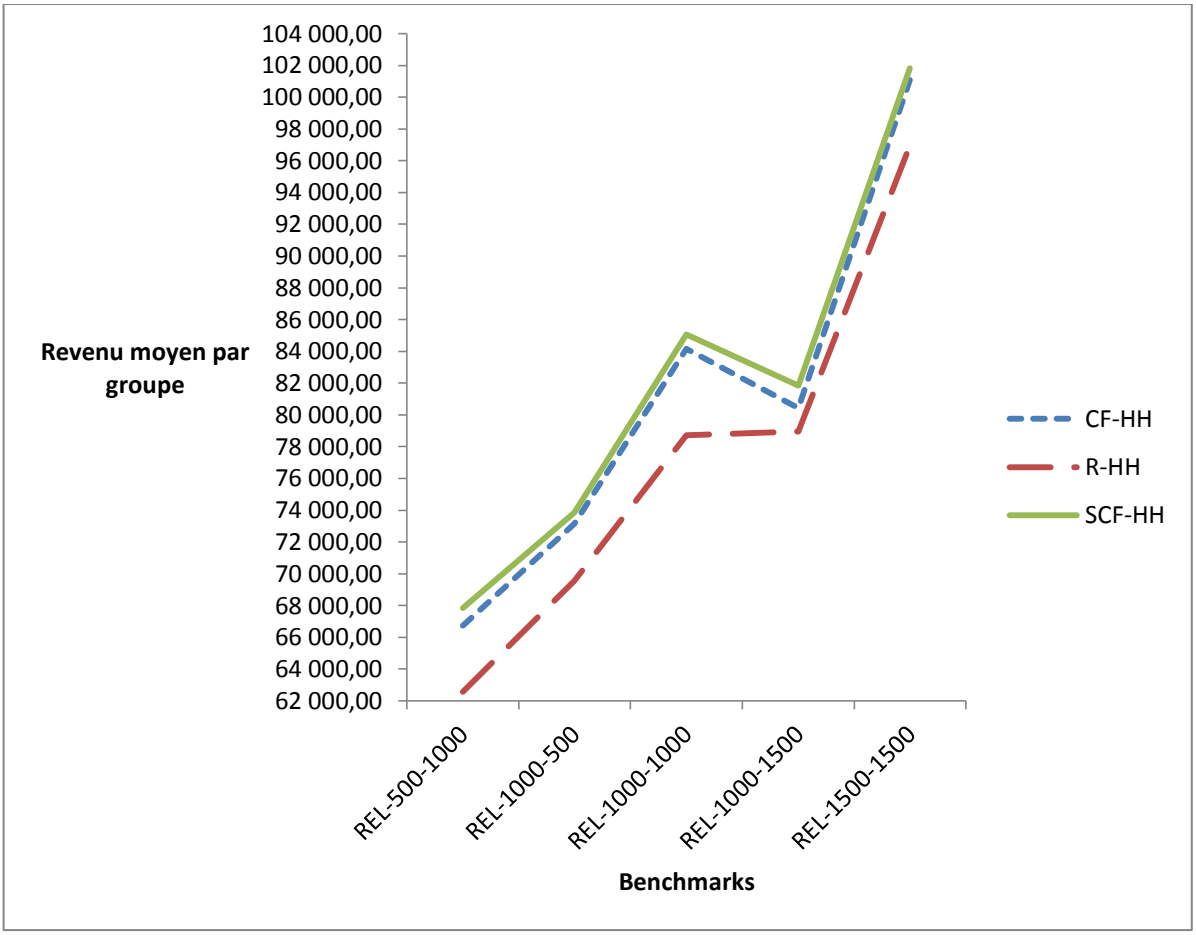

Figure 17. Comparaison des 3 hyper-heuristiques par rapport à la qualité de la solution pour chaque groupe d'instances

La combinaison d'une approche informée - cherchant à sélectionner une heuristique de bas niveau la plus méritante- avec une légère perturbation -caractérisée par le fait de choisir une heuristique de bas niveau aléatoirement et ceci d'après une faible probabilité- offre une possibilité de trouver de bons résultats en un temps raisonnable, comme le montrent la figure 17 et la figure 18, où l'hyper-heuristique SCF-HH donne des résultats encourageants par rapport à CF-HH et R-HH. Vu que SCF-HH s'impose en général dans les 5 groupes d'instances, ses performances devraient être comparées à une méthode classique afin de l'évaluer et ainsi prouver l'efficacité d'une telle méthode pour le problème du PDG.

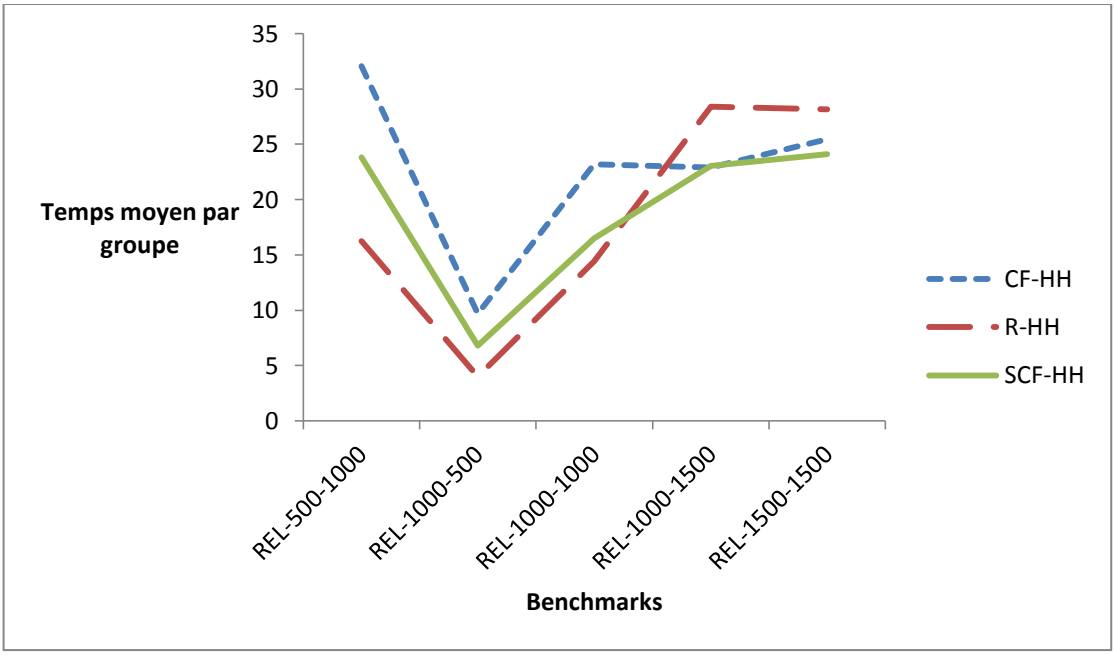

Figure 18. Comparaison du temps d'exécution des 3 hyper-heuristiques pour chaque groupe d'instances

4. ÉTUDE COMPARATIVE

Dans cette partie nous comparons la meilleure méthode hyper-heuristique pour le problème du PDG à savoir SCF-HH avec la méthode du SLS, mais aussi la mauvaise méthode hyper-heuristique, soit R-HH, avec la méthode de recherche locale SLS.

SLS est une méthode qui donne des résultats intéressants, elle a été comparée à plusieurs méthodes, entre autre nous citons la méthode Casanova **[7]**, recherche taboue [**8**], algorithme mémétique[**9**]. SLS a donné de très bons résultats par rapport à Casanova et la recherche taboue, et a pu s'imposer dans certaines instances par rapport à l'algorithme mémétique.

SLS travaille avec un paramètre wp = 0.3 qui représente quel mouvement effectuer pour choisir une offre à intégrer dans la solution, ce paramètre est fixé de manière empirique où plusieurs valeurs ont été testées afin de trouver la meilleure.

Pour ce qui concerne le nombre d'itérations maximum pour son exécution, le paramètre *max_iter_SLS* a été fixé à 1 000 000 d'itérations.

Pour les méthodes hyper-heuristiques, leurs paramétrage est le même que celui que nous avons cité plus haut au début de ce chapitre.

Les résultats présentés pour chaque groupe d'instances sont les 13 premières instances avec les 13 dernières, les résultats des tableaux de 8 à 12 représentent la comparaison entre SCF-HH et SLS. Les résultats des tableaux 13 à 17 représentent la comparaison entre RHH et SLS.

Tableau 8. SCF-HH vs SLS sur quelques instances du groupe REL-500-1000

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
in101	**26,50**	**68 315,63750**	30,33	66 170,61719
in102	**22,64**	**68 428,89453**	30,28	67 797,61719
in103	**22,64**	**68 236,46250**	30,31	66 396,95313
in104	**23,99**	**68 596,57188**	33,70	66 356,24219
in105	**25,27**	**70 111,21172**	34,10	67 486,31250
in106	**22,65**	**66 554,30469**	28,89	65 359,58594
in107	**22,44**	68 587,17188	30,20	**68 618,14063**
in108	**24,84**	**74 506,78750**	31,09	71 529,20313
in109	**22,55**	**66 203,72656**	29,00	66 061,50781
in110	**24,43**	**64 867,79063**	29,76	63 875,10156
in111	**25,22**	**72 969,16406**	31,00	68 741,46094
in112	**24,02**	**67 163,64844**	30,50	65 533,18750
in113	**23,82**	68 115,35626	31,47	**68 542,17188**
in188	**23,09**	**67 662,82070**	30,23	66 321,70313
in189	**27,29**	**68 337,96875**	30,73	64 202,40625
in190	**22,45**	**68 022,94141**	31,95	67 434,95313
in191	**23,87**	**71 236,99375**	30,59	69 246,51563
in192	**23,43**	66 016,80039	30,16	**66 096,28906**
in193	**21,96**	**67 885,93750**	29,23	67 096,09375
in194	**23,99**	**69 687,35703**	30,98	67 690,10938
in195	**23,30**	66 015,40156	30,02	**69 066,85938**
in196	**23,38**	**68 707,62422**	30,56	68 655,61719
in197	**22,38**	**63 307,77031**	29,00	62 824,19531
in198	**25,30**	**69 717,45625**	30,81	66 522,14063
in199	**23,01**	**69 711,90859**	30,06	69 158,37500
in200	**22,72**	71 353,74297	30,86	**71 432,70313**

Le tableau8 nous donne les performances de SCF-HH et SLS sur le groupe **REL-500-1000**. Ce que ces résultats nous montrent c'est que SCF-HH s'impose dans la qualité de la solution dans plusieurs instances, avec une exécution plus rapide dans toutes les instances.

Tableau 9. SCF-HH vs SLS sur quelques instances du groupe REL-1000-1000

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
in201	**16,14**	81 387,33047	19,42	**81 557,74219**
in202	**17,63**	**90 408,93047**	20,94	87 003,51563
in203	**16,44**	**86 239,21094**	20,67	81 696,49219
in204	**16,64**	**82 575,65859**	20,59	81 969,04688
in205	**15,84**	81 622,47656	19,62	**82 676,62500**
in206	**16,17**	**86 436,02579**	19,78	82 427,46094
in207	**17,23**	**91 033,51563**	20,93	91 033,51563
in208	**16,95**	85 173,28906	21,11	**91 782,04688**
in209	**16,09**	86 117,06719	20,05	**86 388,63281**
in210	**15,85**	**85 079,98438**	20,44	85 079,98438
in211	**16,36**	**81 359,82344**	19,06	80 111,96875
in212	**16,39**	81 229,52734	21,04	**81 459,96094**
in213	**16,51**	82 253,71797	20,86	**82 271,00000**
in288	**15,84**	**82 909,68750**	19,11	82 909,68750
in289	**16,71**	80 930,92657	20,02	**82 614,14844**
in290	**16,70**	83 263,24766	20,19	**83 985,00000**
in291	**17,51**	**89 388,06797**	20,59	88 446,52344
in292	**16,92**	89 974,89453	21,45	**90 265,55469**
in293	**16,88**	81 301,78282	21,03	**82 204,57031**
in294	**16,47**	**88 174,37578**	20,78	85 472,31250
in295	**15,78**	**85 770,60938**	20,36	83 086,79688
in296	**15,78**	84 254,82656	19,72	**85 220,38281**
in297	**16,57**	81 542,87891	19,93	**81 580,95313**
in298	**16,24**	**83 847,04297**	20,50	83 389,87500
in299	**17,21**	**89 143,13516**	21,56	84 135,57813
in300	**16,89**	**88 373,32031**	20,65	88 373,32031

Dans certaines instances du groupe **REL-1000-1000**, SLS arrive à prendre un léger avantage dans le profit retourné pour la solution trouvée, mais s'incline dans le reste et n'offre pas d'exécution plus rapide que SCF-HH comme le montre le tableau9.

Tableau 10. SCF-HH vs SLS sur quelques instances du groupe REL-1000-500

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
in401	**6,70**	**75 588,99610**	8,75	72 948,07031
in402	**6,75**	71 693,71953	8,91	**72 820,03125**
in403	**6,73**	**74 843,96094**	9,13	74 843,96094
in404	**6,86**	**78 761,68750**	9,41	78 761,68750
in405	**7,20**	**74 806,79688**	9,27	72 712,39063
in406	**6,47**	**71 791,03125**	9,03	71 791,03125
in407	**7,27**	**73 935,28125**	9,89	73 935,28125
in408	**6,79**	**77 018,83594**	8,19	77 018,83594
in409	**6,28**	**69 102,72266**	8,31	67 420,35156
in410	**7,02**	**71 791,57813**	9,75	71 791,57813
in411	**6,84**	**71 272,47656**	9,17	71 200,55469
in412	**6,17**	**75 292,63281**	9,02	75 292,63281
in413	**6,95**	72 561,90625	9,64	**73 638,23438**
in488	**6,34**	**77 928,14844**	8,89	77 928,14844
in489	**6,56**	**73 740,02969**	8,94	70 407,90625
in490	**6,35**	**71 423,73438**	8,33	71 423,73438
in491	**6,97**	**76 284,35625**	9,52	75 025,49219
in492	**7,63**	72 318,21875	9,47	**72 589,17188**
in493	**6,63**	**76 953,24766**	8,27	76 481,16406
in494	**6,79**	**74 398,20313**	9,17	74 398,20313
in495	**6,49**	**72 919,67969**	8,67	71 911,85938
in496	**7,43**	**73 533,67344**	9,67	73 376,74219
in497	**7,28**	**76 076,26563**	8,86	74 480,69531
in498	**6,32**	74 213,25000	8,10	**80 992,92188**
in499	**6,95**	**72 113,62266**	9,53	71 246,87500
in500	**6,59**	**73 320,69063**	8,92	70 739,92188

Le tableau10 présente les résultats de l'exécution de SCF-HH et SLS appliquées à certaines instances du groupe **REL-1000-500**. Ces résultats nous montrent l'efficacité de l'hyper-heuristique en ce qu'elle offre de meilleurs résultats, plus rapidement que la méthode SLS dans la majorité des instances.

Tableau 11. SCF-HH vs SLS sur quelques instances du groupe REL-1000-1500

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
in501	23,02	**79 864,55937**	**22,85**	79 453,92969
in502	23,02	**80 340,76563**	**22,95**	80 340,76563
in503	**23,57**	**83 277,71094**	24,44	79 010,70313
in504	**22,90**	**81 903,02344**	24,35	81 903,02344
in505	22,97	**77 977,65625**	**22,57**	77 977,65625
in506	21,43	**77 523,67656**	**19,39**	76 722,13281
in507	25,07	**84 174,39063**	**24,68**	77 373,75000
in508	23,36	**82 149,27344**	**22,10**	80 446,06250
in509	**23,49**	**82 672,25781**	24,87	82 672,25781
in510	21,62	**80 102,35000**	**20,26**	78 841,15625
in511	22,10	78 793,83594	**20,26**	**79 754,37500**
in512	22,91	**82 445,79687**	**22,46**	82 342,70313
in513	23,53	**84 079,56250**	**21,73**	74 697,44531
in588	**23,73**	**84 770,49219**	24,37	83 872,46875
in589	**25,04**	**80 834,24219**	26,33	74 174,32031
in590	**23,11**	**79 170,10156**	23,57	78 712,15625
in591	21,56	**80 227,53125**	**19,62**	80 227,53125
in592	**23,05**	**80 131,55547**	23,84	78 381,93750
in593	24,25	**86 411,11719**	**23,52**	80 658,93750
in594	22,42	**86 112,90625**	**22,18**	86 112,90625
in595	23,32	**85 061,31250**	**21,93**	77 582,07813
in596	21,21	**78 967,11563**	**19,38**	77 878,38281
in597	22,54	**80 443,87500**	**21,93**	79 195,67188
in598	**25,05**	**80 272,86328**	28,33	79 201,96875
in599	22,04	**77 527,33594**	**21,17**	77 527,33594
in600	23,73	**91 538,68750**	**21,11**	82 555,91406

Le tableau ci-dessus donne les résultats trouvés par SCF-HH et SLS dans certaines instances du groupe **REL-1000-1500**. On remarque que SCF-HH est plus performante que SLS dans plusieurs instances du groupe avec une exécution plus ou moins proche entre les deux méthodes. Dans certaines instances, SCF-HH et SLS donnent les mêmes résultats.

Tableau 12. SCF-HH vs SLS sur quelques instances du groupe REL-1500-1500

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
in601	24,52	**105 402,77266**	**24,13**	99 044,32813
in602	**23,51**	95 328,21875	24,48	**98 164,23438**
in603	**23,49**	**95 039,86719**	24,27	94 126,96094
in604	**23,35**	**103 568,86719**	24,20	103 568,86719
in605	**24,73**	101 470,54688	27,08	**102 404,76563**
in606	**23,68**	104 346,07031	24,26	**105 218,21094**
in607	**23,93**	**105 869,44531**	25,01	105 869,44531
in608	23,83	**101 824,13281**	**23,68**	94 948,61719
in609	**23,85**	**99 023,57031**	25,07	98 566,94531
in610	27,08	**111 835,92188**	**26,82**	102 468,60156
in611	**23,48**	**98 826,87032**	24,38	98 472,89063
in612	**25,29**	**106 056,07813**	25,91	104 749,68750
in613	**24,00**	93 371,29063	26,13	**94 605,78125**
in688	**24,34**	**103 959,20938**	25,41	103 742,53125
in689	**24,78**	**101 767,98437**	27,10	101 765,07031
in690	**23,89**	**103 094,79531**	24,79	102 312,40625
in691	23,61	**102 167,67969**	**23,21**	102 167,67969
in692	**24,50**	98 865,29688	26,07	**99 829,69531**
in693	**23,89**	**99 616,92969**	25,08	99 616,92969
in694	**23,24**	**108 114,12500**	24,69	108 114,12500
in695	**25,55**	**102 094,31406**	26,19	99 678,63281
in696	**23,38**	97 351,56251	24,04	**98 035,48438**
in697	**22,74**	**97 995,98438**	23,42	97 738,42188
in698	**24,29**	**101 554,40313**	24,91	100 076,42969
in699	**24,46**	**103 762,70313**	26,63	103 762,70313
in700	**24,40**	97 675,50781	25,76	**101 510,20313**

Le tableau12 présente les résultats donnés par SCF-HH et SLS appliquées au groupe **REL-1500-1500**. Ces résultats nous montrent que SCF-HH s'impose en termes de qualité de la solution trouvée et avec une exécution plus rapide que SLS dans plusieurs instances du groupe.

Tableau 13. R-HH vs SLS sur quelques instances du groupe REL-500-1000

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
in101	**18,18**	**68 619,06250**	30,33	66 170,61719
in102	**15,54**	64 800,01563	30,28	**67 797,61719**
in103	**15,50**	**68 983,25000**	30,31	66 396,95313
in104	**14,05**	61 949,51563	33,70	**66 356,24219**
in105	**17,33**	64 438,91406	34,10	**67 486,31250**
in106	**15,83**	65 117,76953	28,89	**65 359,58594**
in107	**15,22**	59 622,38281	30,20	**68 618,14063**
in108	**16,89**	64 000,28906	31,09	**71 529,20313**
in109	**16,24**	59 982,78516	29,00	**66 061,50781**
in110	**16,78**	63 566,44922	29,76	**63 875,10156**
in111	**16,52**	**72 969,16406**	31,00	68 741,46094
in112	**15,83**	**65 590,26563**	30,50	65 533,18750
in113	**16,33**	64 122,30469	31,47	**68 542,17188**
in188	**17,21**	61 769,67188	30,23	**66 321,70313**
in189	**17,01**	63 346,44922	30,73	**64 202,40625**
in190	**16,67**	61 861,35938	31,95	**67 434,95313**
in191	**16,62**	61 434,90234	30,59	**69 246,51563**
in192	**14,89**	57 668,03125	30,16	**66 096,28906**
in193	**15,68**	60 857,00781	29,23	**67 096,09375**
in194	**16,89**	62 808,57031	30,98	**67 690,10938**
in195	**15,99**	61 907,14844	30,02	**69 066,85938**
in196	**16,47**	60 672,15625	30,56	**68 655,61719**
in197	**15,31**	56 949,31250	29,00	**62 824,19531**
in198	**16,52**	66 144,22656	30,81	**66 522,14063**
in199	**16,28**	62 216,54688	30,06	**69 158,37500**
in200	**15,32**	66 243,24219	30,86	**71 432,70313**

D'après les résultats trouvés dans le tableau13, la qualité de la solution trouvée par SLS est plus performante que celle de R-HH dans la majorité des instances pour le groupe **REL-500-1000**, soit dit en passant, R-HH s'exécute plus rapidement que SLS, et offre dans très peu d'instances de très bons résultats.

Tableau 14. R-HH vs SLS sur quelques instances du groupe REL-1000-1000

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
in201	**14,34**	70 388,03125	19,42	**81 557,74219**
in202	**13,85**	76 900,35938	20,94	**87 003,51563**
in203	**14,09**	77 169,17188	20,67	**81 696,49219**
in204	**14,21**	79 241,41406	20,59	**81 969,04688**
in205	**14,01**	72 614,98438	19,62	**82 676,62500**
in206	**14,25**	76 224,60156	19,78	**82 427,46094**
in207	**14,13**	89 048,46875	20,93	**91 033,51563**
in208	**14,20**	85 173,28906	21,11	**91 782,04688**
in209	**14,35**	78 044,66406	20,05	**86 388,63281**
in210	**13,99**	82 949,44531	20,44	**85 079,98438**
in211	**14,52**	**80 752,77344**	19,06	80 111,96875
in212	**14,66**	78 715,03906	21,04	**81 459,96094**
in213	**15,58**	73 899,15625	20,86	**82 271,00000**
in288	**14,45**	78 303,71875	19,11	**82 909,68750**
in289	**15,49**	76 246,18750	20,02	**82 614,14844**
in290	**13,45**	77 931,26563	20,19	**83 985,00000**
in291	**14,74**	80 111,68750	20,59	**88 446,52344**
in292	**14,87**	84 571,60938	21,45	**90 265,55469**
in293	**14,78**	76 516,00781	21,03	**82 204,57031**
in294	**14,34**	77 175,76563	20,78	**85 472,31250**
in295	**13,66**	75 693,48438	20,36	**83 086,79688**
in296	**14,55**	79 936,94531	19,72	**85 220,38281**
in297	**14,13**	76 160,31250	19,93	**81 580,95313**
in298	**12,98**	81 654,10156	20,50	**83 389,87500**
in299	**14,16**	76 252,15625	21,56	**84 135,57813**
in300	**14,74**	79 439,40625	20,65	**88 373,32031**

Pour le groupe d'instances **REL-1000-1000**, SLS s'impose très nettement par rapport à R-HH dans la qualité de la solution retournée, comme le montre le tableau14 ; ainsi la rapidité d'exécution de R-HH ne lui a pas permis d'atteindre des résultats plus intéressants.

Tableau 15. R-HH vs SLS sur quelques instances du groupe REL-1000-500

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
in401	**3,87**	68 009,35938	8,75	**72 948,07031**
in402	**3,98**	71 454,78906	8,91	**72 820,03125**
in403	**3,69**	67 873,03125	9,13	**74 843,96094**
in404	**3,94**	69 322,72656	9,41	**78 761,68750**
in405	**4,02**	**74 138,28125**	9,27	72 712,39063
in406	**4,21**	71 009,36719	9,03	**71 791,03125**
in407	**3,94**	69 484,14844	9,89	**73 935,28125**
in408	**3,86**	69 729,71094	8,19	**77 018,83594**
in409	**3,61**	65 251,77344	8,31	**67 420,35156**
in410	**3,90**	**71 791,57813**	9,75	71 791,57813
in411	**4,20**	**71 272,47656**	9,17	71 200,55469
in412	**3,66**	**75 292,63281**	9,02	75 292,63281
in413	**3,79**	72 561,90625	9,64	**73 638,23438**
in488	**3,86**	70 299,06250	8,89	**77 928,14844**
in489	**3,64**	67 364,98438	8,94	**70 407,90625**
in490	**3,80**	63 984,82031	8,33	**71 423,73438**
in491	**4,20**	71 905,09375	9,52	**75 025,49219**
in492	**4,10**	70 484,99219	9,47	**72 589,17188**
in493	**3,80**	76 170,53125	8,27	**76 481,16406**
in494	**4,25**	66 717,17188	9,17	**74 398,20313**
in495	**3,88**	**72 919,67969**	8,67	71 911,85938
in496	**4,24**	**73 376,74219**	9,67	73 376,74219
in497	**4,41**	72 196,35156	8,86	**74 480,69531**
in498	**4,30**	67 302,00781	8,10	**80 992,92188**
in499	**4,22**	63 359,17188	9,53	**71 246,87500**
in500	**3,92**	66 948,61719	8,92	**70 739,92188**

R-HH améliore très légèrement la solution dans très peu d'instances du groupe **REL-1000-1500** avec une exécution plus rapide dans toutes les instances du groupe par rapport à SLS, comme le montre le tableau15; ainsi dans la plus grande majorité des instances, SLS offre une qualité de solution plus importante.

Tableau 16. R-HH vs SLS sur quelques instances du groupe REL-1000-1500

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
in501	29,08	77 738,97656	**22,85**	**79 453,92969**
in502	28,87	76 348,53906	**22,95**	**80 340,76563**
in503	28,13	**79 640,48438**	**24,44**	79 010,70313
in504	27,54	77 668,24219	**24,35**	**81 903,02344**
in505	27,33	**77 977,65625**	**22,57**	77 977,65625
in506	28,05	**77 254,13281**	**19,39**	76 722,13281
in507	29,36	**84 174,39063**	**24,68**	77 373,75000
in508	28,43	**82 149,27344**	**22,10**	80 446,06250
in509	30,13	76 472,35156	**24,87**	**82 672,25781**
in510	27,19	74 997,00000	**20,26**	**78 841,15625**
in511	26,90	78 793,83594	**20,26**	**79 754,37500**
in512	28,89	78 998,21094	**22,46**	**82 342,70313**
in513	29,47	**84 079,56250**	**21,73**	74 697,44531
in588	28,56	**85 201,01563**	**24,37**	83 872,46875
in589	29,03	**80 834,24219**	**26,33**	74 174,32031
in590	28,02	**79 170,10156**	**23,57**	78 712,15625
in591	26,92	77 586,61719	**19,62**	**80 227,53125**
in592	27,77	76 314,15625	**23,84**	**78 381,93750**
in593	31,16	**81 859,21875**	**23,52**	80 658,93750
in594	28,86	80 210,50000	**22,18**	**86 112,90625**
in595	28,64	77 402,39063	**21,93**	**77 582,07813**
in596	27,80	73 977,00781	**19,38**	**77 878,38281**
in597	27,32	77 642,75781	**21,93**	**79 195,67188**
in598	26,66	75 883,96875	**28,33**	**79 201,96875**
in599	27,90	76 078,31250	**21,17**	**77 527,33594**
in600	29,54	**91 538,68750**	**21,11**	82 555,91406

Pour le groupe **REL-1000-1500,** les résultats nous montrent que dans certaines instances SLS s'impose sur R-HH, et dans d'autres R-HH s'impose sur SLS, comme nous l'indique le tableau16. Nous remarquons aussi la rapidité d'exécution de SLS dans ce groupe.

Tableau 17. R-HH vs SLS sur quelques instances du groupe REL-1500-1500

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
in601	27,82	97 068,27344	**24,13**	**99 044,32813**
in602	27,16	95 328,21875	**24,48**	**98 164,23438**
in603	28,31	89 086,69531	**24,27**	**94 126,96094**
in604	28,53	103 289,63281	**24,20**	**103 568,86719**
in605	27,11	96 930,74219	**27,08**	**102 404,76563**
in606	27,61	96 238,88281	**24,26**	**105 218,21094**
in607	27,85	102 119,13281	**25,01**	**105 869,44531**
in608	27,92	**94 949,40625**	**23,68**	94 948,61719
in609	27,98	**99 023,57031**	**25,07**	98 566,94531
in610	30,18	101 089,13281	**26,82**	**102 468,60156**
in611	26,89	90 639,53906	**24,38**	**98 472,89063**
in612	28,18	94 273,08594	**25,91**	**104 749,68750**
in613	28,44	90 353,45313	**26,13**	**94 605,78125**
in688	27,07	99 796,36719	**25,41**	**103 742,53125**
in689	28,19	92 270,11719	**27,10**	**101 765,07031**
in690	27,87	96 793,16406	**24,79**	**102 312,40625**
in691	28,11	**102 167,67969**	**23,21**	102 167,67969
in692	28,32	98 865,29688	**26,07**	**99 829,69531**
in693	28,19	98 747,82813	**25,08**	**99 616,92969**
in694	27,84	96 316,60938	**24,69**	**108 114,12500**
in695	29,45	**101 671,88281**	**26,19**	99 678,63281
in696	27,76	95 002,00781	**24,04**	**98 035,48438**
in697	27,42	94 195,17188	**23,42**	**97 738,42188**
in698	28,05	92 468,02344	**24,91**	**100 076,42969**
in699	26,78	92 010,72656	**26,63**	**103 762,70313**
in700	27,67	90 723,92969	**25,76**	**101 510,20313**

Le tableau 17 nous présente les résultats de la méthode SLS et l'hyper-heuristique R-HH appliquées au groupe d'instances **REL-1500-1500**. Ces résultats nous montrent l'efficacité de SLS sur ce groupe en termes de qualité de la solution ainsi que du temps d'exécution par rapport à R-HH qui nous donne dans la plus part du temps des résultats moyens.

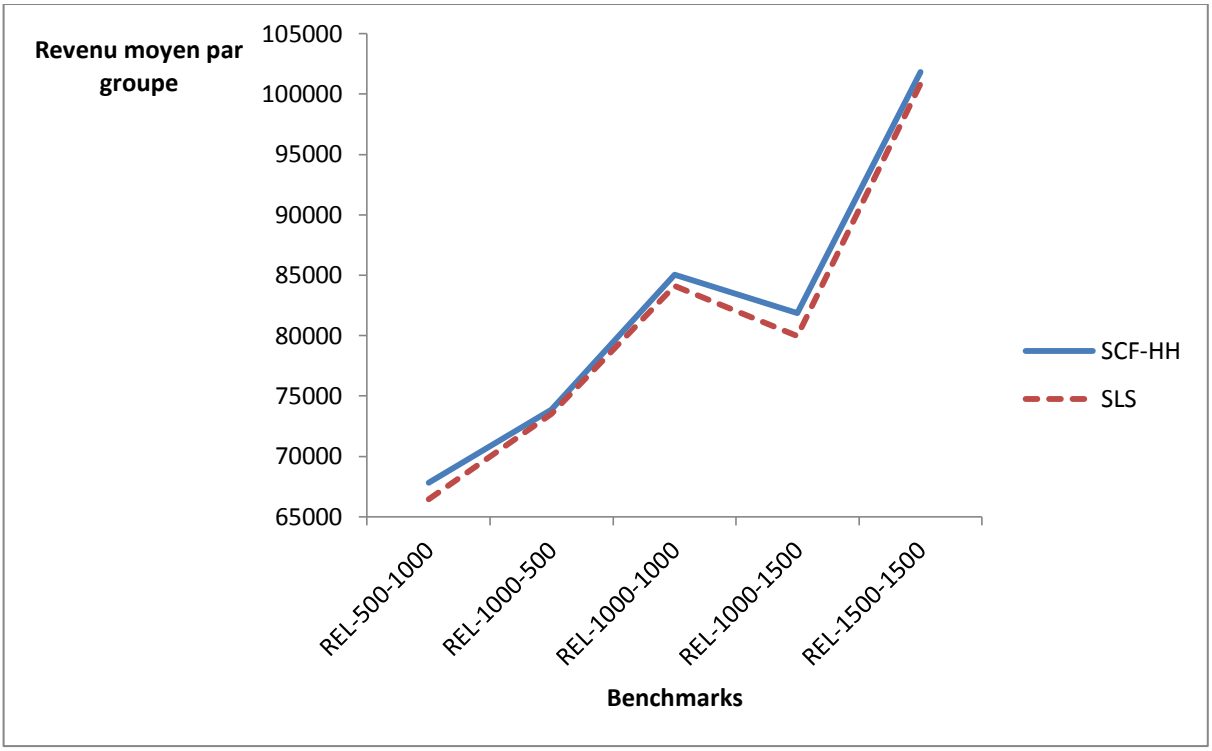

Figure 19. Comparaison entre SLS et SCF-HH par rapport à la qualité de la solution pour chaque groupe d'instances

Les résultats moyens de SCF-HH et SLS pour chaque groupe d'instances sont présentés dans les figures 19 et 20, où en moyenne SCF-HH offre des solutions de qualité de manière assez rapide par rapport à SLS, comme illustré dans les graphes des figures.

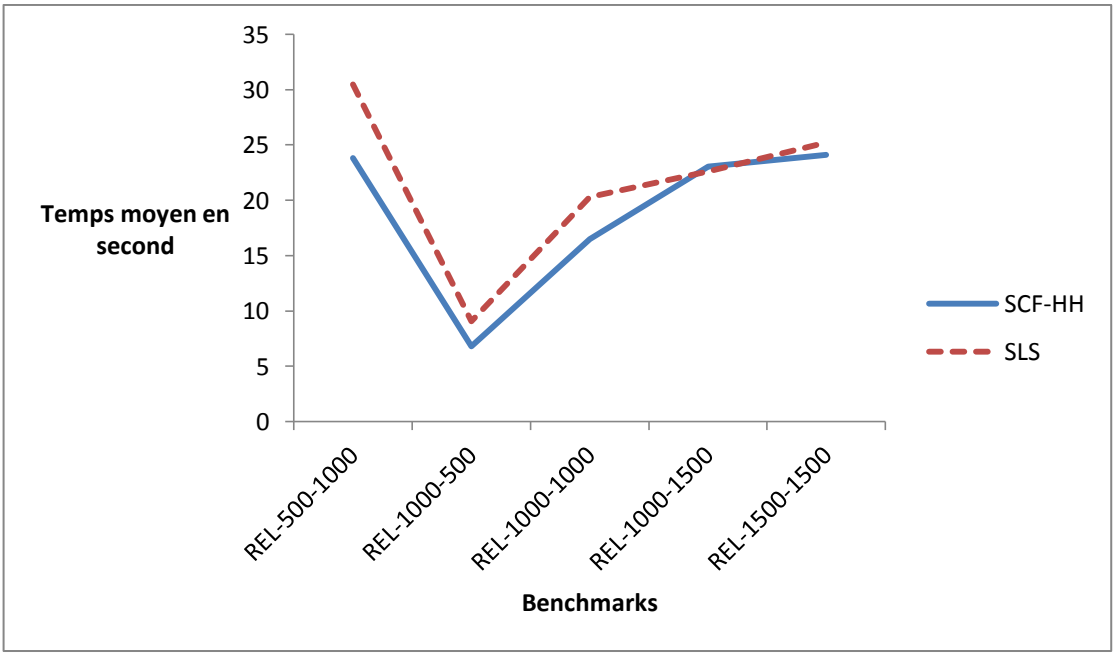

Figure 20. Graphe comparatif entre SLS et SCF-HH par rapport au temps d'execution moyen pour chaque groupe d'instances

Tableau 18. Comparaison entre SCF-HH et SLS sur la moyenne des 5 groupes d'instances du problème

Instances	SCF-HH		SLS	
	Temps	Solution	Temps	Solution
REL-500-1000	**23,82**	**67 826,4412**	30,49	66 460,9956
REL-1000-500	**6,79**	**73 859,7833**	9,07	73 541,0609
REL-1000-1000	**16,50**	**85 065,3156**	20,28	84 108,9606
REL-1000-1500	23,06	**81 843,9000**	**22,61**	79 975,6916
REL-1500-1500	**24,12**	**101 804,8757**	25,20	100 828,2766

Les tableaux 18 et 19 présentent le temps d'exécution moyen avec le profit moyen des différents groupes pour chaque méthode. Le tableau 18 nous montre la supériorité de la méthode SCF-HH par rapport à SLS, sauf pour le temps d'exécution du groupe **REL-1000-1500** où SLS s'impose avec **0,45 seconde**, ce qui démontre l'efficacité de l'hyper-heuristique SCF-HH pour le problème du PDG. Ces résultats peuvent être justifiés par le fait que SCF-HH exploite efficacement l'ensemble des heuristiques de bas niveau, rendant ainsi possible l'exploration de nouvelle zone de l'espace de recherche grâce à la diversité de ces méthodes et à un appel intelligent vers celles-ci.

Tableau 19. Comparaison de la moyenne des 5 groupes d'instances entre R-HH et SLS

Instances	R-HH		SLS	
	Temps	Solution	Temps	Solution
REL-500-1000	**16,22**	62 563,9407	30,49	**66 460,9956**
REL-1000-500	**4,02**	69 562,4518	9,07	**73 541,0609**
REL-1000-1000	**14,40**	78 728,6097	20,28	**84 108,9606**
REL-1000-1500	28,40	78 961,0330	**22,61**	**79 975,6916**
REL-1500-1500	28,16	96 939,0750	**25,20**	**100 828,2766**

Pour le Tableau 19, R-HH ne s'impose que dans le temps d'exécution pour trois groupes seulement qui sont REL-500-1000, REL-1000-500 et REL-1000-1000. Pour le reste, SLS est de loin plus efficace en temps et en qualité de solution. Ce qui a pénalisé R-HH c'est sa naïveté dans la sélection des heuristiques à exécuter pour la prochaine itération, même si dans certaines instances R-HH offre de très bons résultats, comparé à SLS qui cherche à travailler de manière stable et efficace.

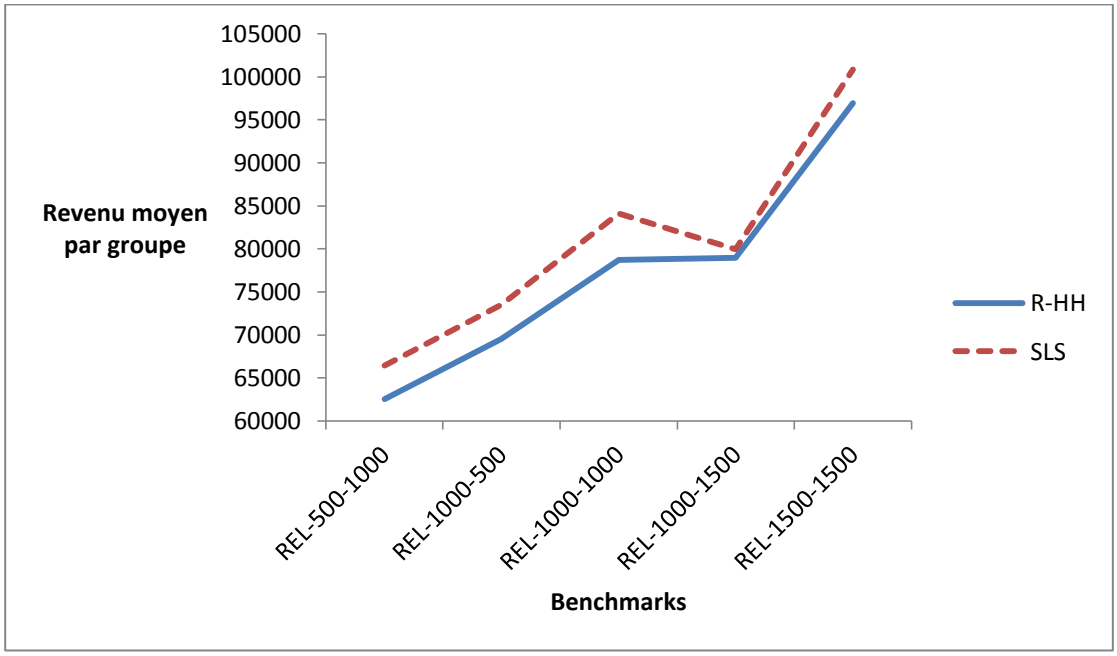

Figure 21. Comparaison de la qualité de la solution retourné par SLS et R-HH

Les figures 21 et 22 schématisent les résultats donnés par l'hyper-heuristique aléatoire R-HH et SLS, et montrent avec clarté la nette supériorité de SLS.

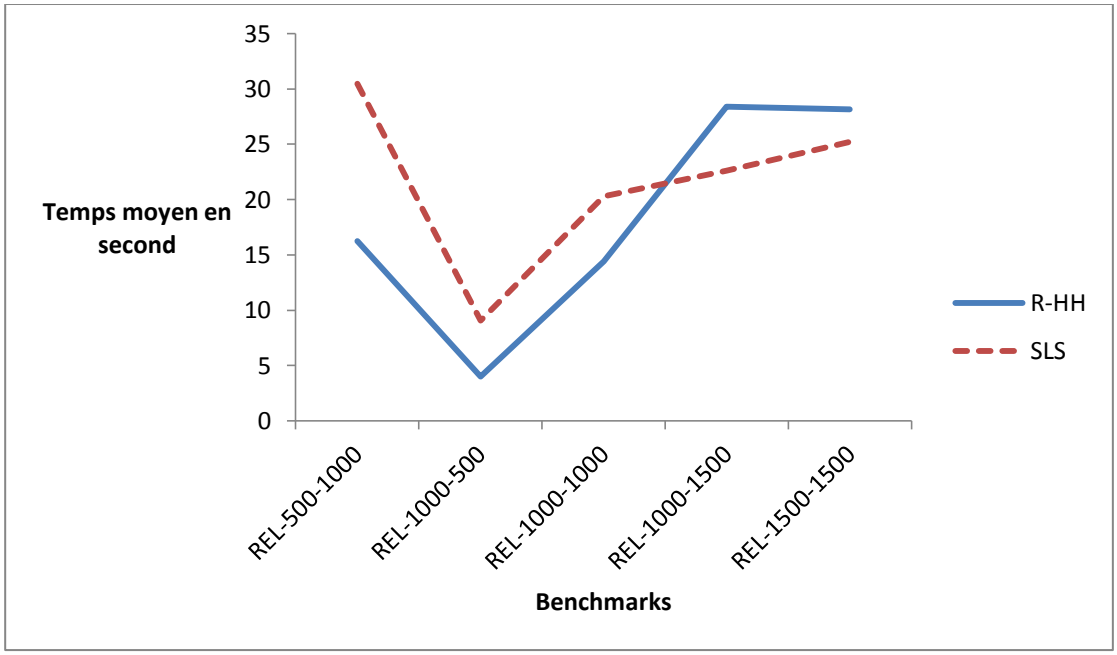

Figure 22. Graphe comparatif entre SLS et R-HH sur le temps d'exécution moyen

5. CONCLUSION

L'hyper-heuristique constitue une nouvelle approche de résolution de problème en faisant coexister plusieurs méthodes de résolution propre au problème dans un but commun, celui de trouver la meilleure solution approchée et éventuellement l'optimale.

Pour le problème du PDG, des méthodes classiques ont été utilisées dans le but de le résoudre. Cette fois-ci une méthode de haut niveau essaie de choisir à une itération donnée une heuristique afin d'améliorer la solution courante, ce choix peut être aveugle comme il peut utiliser des méthodes avancées pour porter certains jugements sur ces méthodes.

L'utilisation aveugle de méthodes peut être efficace dans certain cas, mais les résultats obtenus nous montrent qu'en moyenne cette façon de procéder donne de moins bons résultats qu'une méthode métaheuristique classique. Cependant, en exploitant les performances de ces méthodes de bas niveau par une technique de sélection plus élaborée, nous pouvons produire des résultats intéressants améliorant légèrement la solution en comparant avec une méthode classique. Cela dit, une méthode de sélection informée peut être encore plus efficace si de temps en temps nous ajoutions une perturbation qui peut perturber l'ordre de choix. En effet, si une

méthode qui n'était pas censée être appelée, le soit par le fruit du hasard, elle peut apporter le plus tant attendu dans le processus de recherche.

CONCLUSION GÉNÉRALE

Les métaheuristiques forment une famille d'algorithmes d'optimisation souvent très efficaces pour plusieurs problèmes, mais tant que nous ne trouvons pas la solution optimale d'un problème d'optimisation il faut essayer de voir plus loin dans leurs résolutions, de concevoir une nouvelle façon de procéder, une nouvelle plateforme de résolution, une méthode de haut niveau d'abstraction jouant le rôle de chef pour un ensemble de méthodes (heuristique et/ou métaheuristique) du problème. C'est pour cette raison que les hyper-heuristiques ont fait leur apparition, afin de pouvoir exploiter efficacement les forces de ces méthodes et d'éviter de tomber dans certains pièges dus à leurs faiblesses. La structure modulaire des hyper-heuristiques fait aussi leur force, car nous pouvons changer la méthode (ou le module) de sélection d'heuristique sans pour autant changer la structure de la méthode (ou du module) d'acceptation des solutions, et vice-versa.

Ces hyper-heuristiques ont été appliquées à plusieurs problèmes et ont donné de bons résultats, par exemple le problème d'emploi du temps universitaire (*university timetabling*) [**12**].

C'est à partir de ces différents points qu'est venu l'envie d'aborder le problème du PDG à l'aide de méthodes hyper-heuristiques, et d'explorer certains concepts de sélection d'heuristiques de bas niveau, à savoir : décider quelle méthode de sélection est la plus efficace, entre une méthode informée, une méthode non-informée et une méthode hybride, et de les comparer par la suite avec une métaheuristique efficace, à savoir la recherche locale SLS.

Un point important que nous montre le chapitre 4 avec les divers résultats sur les trois types d'hyper-heuristiques développées et la comparaison avec la recherche locale SLS, est l'importance de la stratégie de sélection des heuristiques de bas niveau. Avec une méthode non-informé (aléatoire) nous ne pouvons pas espérer produire d'excellents résultats dans tous les cas, vu la naïveté et la simplicité de cette méthode, par contre une méthode informée qui accepte de temps en temps un choix

aléatoire peut donner de biens meilleurs résultats en temps et en qualité de la solution qu'une méthode purement informée. Grâce à cette perturbation engendrée, l'hyper-heuristique peut explorer de nouvelles zones de l'espace de recherche la rapprochant ainsi de l'optimum global.

Nous envisageons pour nos travaux futurs :

- Essayer d'hybrider d'autres méthodes de sélection d'heuristiques de bas niveau.
- Appliquer une hyper-heuristique génératrice pour générer une nouvelle méthode heuristique pour le problème du PDG.
- Explorer d'autres méthodes d'acceptation de solution, en acceptant certaines dégradations de la solution renvoyée par les heuristiques de bas niveau.
- Etudier l'efficacité de l'hyper-heuristique si nous augmentons ou diminuons le nombre d'heuristiques de bas niveau.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Andersson A, Tenhunen M, Ygge, F. « ***Integer programming for combinatorial auction winner determination*** ». *In Proceedings of Fourth International Conference on MultiAgent Systems*. IEEE, 2000. pp : 39 - 46.

[2] Bai R. « ***An investigation of novel approaches for optimizing retail shelf space allocation*** ». Thèse de doctorat. University of Nottingham, 2005.

[3] Ben-Ameur H, « ***Enchères multi-objets pour la négociation automatique et le commerce électronique »***, mémoire pour l'obtention du grade de maître en sciences. Université de Laval en France, Avril 2001.

[4] Boughaci D, «APPROCHES METAHEURISTIQUES POUR LES PROBLEMES DIFFICILES (MAX-SAT et PDG) », thèse de doctorat. Université USTHB, Février 2008.

[5] Boughaci D, « ***A Differential Evolution Algorithm for the Winner Determination Problem in Combinatorial Auctions*** ». *In Electronic Notes in Discrete Mathematics*, 2010. vol. 36, pp: 535 - 542.

[6] Boughaci D, Benhamou B, Drias H, « ***Stochastic local search for the optimal winner determination problem in combinatorial auctions*** ». *In Principles and Practice of Constraint Programming*. Springer Berlin/Heidelberg, 2008. pp: 593 - 597.

[7] Boughaci D, Benhamou B, Drias H, « ***Une recherche locale stochastique pour le problème de la détermination du gagnant dans les enchères combinatoires*** ». *In JFPC 2008-Quatrièmes Journées Francophones de Programmation par Contraintes*. 2008. pp : 59-68.

[8] Boughaci D, Benhamou B, Drias H, « ***Local Search Methods for the Optimal Winner Determination Problem in Combinatorial Auctions*** ». *In Journal of Mathematical Modelling and Algorithms*, 2010, vol. 9, pp: 165 - 180.

[9] Boughaci D, Benhamou B, Drias H, « ***A memetic algorithm for the optimal winner determination problem*** ». *In Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 2009. vol. 13, pp: 905 - 917.

[10] Burke E.K, Curtois T, Hyde M, Kendall G, Ochoa G, Petrovic S, Vasquez-Rodriguez J.A, Gendreau M, « ***Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms*** ». *In IEEE Congress on Evolutionary Computation* (CEC 2010). 2010. pp. 3073 - 3080.

[11] Burke E.K, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward J.R, « ***A classification of hyper-heuristic approaches*** ». *Handbook of Metaheuristics*, 2010, pp : 449 - 468.

[12] Burke E.K, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward J.R, « ***Hyper-heuristics: A survey of the state of the art*** ». *In Computer Science Tech. Rep. NOTTCS-TR-SUB-0906241418--2747*, School of *Computer Science and Information Technology, University of Nottingham*, 2010.

[13] Burke E.K, Hyde M.R, Kendall G, Ochoa G, Ozcan E, Woodward J.R, « ***Exploring hyper-heuristic methodologies with genetic programming*** ». *In Computational Intelligence*, 2009, pp: 177 - 201.

[14] Burke E.K, Kendall G, Misir M, Ozcan E, « ***Monte carlo hyper-heuristics for examination timetabling*** ». *Annals of Operations Research*, 2012, pp: 1 - 18.

[15] Chakhlevitch K, Cowling P. « ***Hyperheuristics: recent developments*** ». *Adaptive and Multilevel Metaheuristics*, Springer-Verlag Berlin Heidelberg, 2008, pp: 3 - 29.

[16] Chen P.Y, Wang D.W, « ***Genetic algorithm for solving winner determination in combinatorial auctions***». *JOURNAL-NORTHEASTERN UNIVERSITY NATURAL SCIENCE*, 2003. pp: 7 -10.

[17] Cowling P, Kendall G, Soubeiga E. « ***A hyperheuristic approach to scheduling a sales summit*** ». *Practice and Theory of Automated Timetabling III*, 2001, pp: 176 - 190.

[18] Cramton P, Shoham Y, Steinberg R, « ***Combinatorial auctions*** », MIT Press, 2006.

[19] Dréo J, « ***Adaptation de la méthode des colonies de fourmis pour l'optimisation en variable continues. Application en génie biomédical*** », thèse de doctorat, Université de Val de Marne-Paris 12. Décembre 2004.

[20] Farzi S, « ***Discrete Quantum-Behaved Particle Swarm Optimization for the Multi-Unit Combinatorial Auction Winner Determination Problem*** », *Journal of Applied Sciences*, 2010.

[21] Fujishima Y, Leyton-Brown K, Shoham Y, « ***Taming the computational complexity of combinatorial auctions : optimal and approximate approaches*** », *16th international joint conference on artificial intelligence*, 1999, pp: 48 – 53.

[22] Gan R, Guo Q, Chang H, Yi Y, « ***Ant Colony Optimization for Winner Determination in Combinatorial Auctions*** ». *In Natural Computation*, ICNC 2007. *Third International Conference*, IEEE. 2007. Vol. 4, pp. 441-445.

[23] Gardeux V, « ***Conception d'heuristique d'optimisation pour les problèmes de grande dimension. Application à l'analyse de données de puces à ADN.*** », thèse de doctorat, Université Paris-EST Créteil, Novembre 2011.

[24] Glover F. « ***Tabu search—part I.*** », *ORSA Journal on computing* 1.3, 1989, pp: 190-206.

[25] Guo Y, Lim A, Rodrigues B, Zhu Y. « ***Heuristics for a bidding problem*** », *In Computers & operations research*, 2006. vol. 33, no 8, pp: 2179 - 2188.

[26] Hao J.K, Galinier P, Habib M, « ***Métaheuristique pour l'optimisation combinatoire et l'affectation sous contraintes*** », *Revue d'Intelligence Artificielle*, 1999, pp : 283 – 324.

[27] Hoos HH, Boutilier C, « ***Solving combinatorial auctions using stochastic local search*** », *In proceeding of the 17th national conference on artificial intelligence*, 2000. pp: 22 – 29.

[28] Kendall G, Hussin N. « ***A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology*** ». *Practice and Theory of Automated Timetabling V*, Springer, 2005, pp: 270 - 293.

[29] Kendall G, Mohamad M, « ***Channel assignment optimisation using a hyper-heuristic*** ». *In IEEE Conference on Cybernetics and Intelligent Systems*, 2004. pp: 791 - 796.

[30] Kendall G, Soubeiga E, Cowling P. « ***Choice function and random hyperheuristics*** », *In Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning*, SEAL. 2002. pp: 667 - 671.

[31] Lau H.C, Goh Y.G, « ***An intelligent brokering system to support multi-agent web-based 4th-party logistics*** », *In proceedings of the 14th international conference on tools with artificial intelligence*, 2002, pp: 54

– 61.

[32] Leyton-Brown K, « ***Ressource Allocation in Competitive Multiagent Systems*** », *PhD thesis*, Stanford University, Aout 2003.

[33] Leyton-Brown K, Shoham Y, Tennenholtz M, « ***An algorithm for multi-unit combinatorial auctions*** ». *In proceedings of the national conference on artificial intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000. pp: 56-61.

[34] Meignan D, « ***Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques*** », thèse de doctorat, Université de technologie de Belfort-Montbéliard, Décembre 2008.

[35] Moscato P. « ***On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms.*** », *In: Caltech concurrent computation program, C3P Report* 826. 1989.

[36] Mosteghanemi H, « ***Les essaims d'abeilles pour la recherche d'information à grande échelle.*** », mémoire de magister, Université de Bab-Ezzouar (USTHB), 2011.

[37] Muñoz V, Murillo J, « ***CABRO: Winner Determination Algorithm for Single-unit Combinatorial Auctions*** », *In the proceeding of the 11th International Conference of the Catalan Association for Artificial Intelligence*, 2008, pp: 303 – 312.

[38] Nisan N,« ***Bidding and allocation in combinatorial auctions*** », *In proceedings of ACM Conference on Electronic Commerce*, October 2000, pp: 1 – 12.

[39] Ozcan E, Bykov Y, Birben M, Burke E.K, « ***Examination timetabling using late acceptance hyper-heuristics*** ». *In IEEE Congress on*

Evolutionary Computation (CEC'09), 2009. pp: 997 - 1004.

[40] Ozcan E, Misir M, Ochoa G, Burke E.K, « ***A reinforcement learning-great-deluge hyper-heuristic for examination timetabling*** ». *In International Journal of Applied Metaheuristic Computing* (IJAMC), 2010, vol. 1, no 1, pp: 39 - 59.

[41] Robilliard D, « ***Recherche opérationnelle et optimisation*** », Université Littoral- Côte d'opale, 2010.

[42] Rodriguez J.A.V, Petrovic S, Salhi A. « ***An investigation of hyper-heuristic search spaces*** ». *In IEEE Congress on Evolutionary Computation* (CEC 2007). 2007. pp: 3776 - 3783.

[43] Rothkopf M.H, Pekee A, Ronald M, « ***Computationally manageable combinatorial auctions*** », In Management Science, Vol. 44, No. 8, pp: 1131 - 1147.

[44] Sandholm T, « ***Algorithms for Optimal Winner Determination in Combinatorial Auctions*** », *In Artificial Intelligence*, 1999, pp: 1 - 54.

[45] Sandholm T, Suri S, « ***Improved Optimal Algorithm for Combinatorial Auctions and Generalizations*** », *In Proceeding of the 17th national conference on artificial intelligence*, 2000, pp: 90 - 97.

[46] Sandholm T, Suri S, Gilpin A, Levine D, « ***CABoB: a fast optimal algorithm for combinatorial auctions*** », *In Proceeding of the international joint conference on artificial intelligence*, 2001, pp: 1102 - 1108.

[47] Sin E.S, « ***Reinforcement learning with EGD based hyper heuristic system for exam timetabling problem*** ». *In IEEE International Conference on Cloud Computing and Intelligence Systems* (CCIS), 2011. pp: 462 -

466.

[48] Sirenko S. « ***Classification of Heuristic methods in Combinatorial Optimization*** ». *In International Journal of INFORMATION THEORIES & APPLICATIONS*, 2009, pp: 303 - 322.

[49] Soubeiga E, « ***Developement and application of hyperheuristics to personnel scheduling*** », PhD thesis. Université de Nottingham, Juin 2003.

[50] Storn R, Price K, « ***Minimizing the real functions of the ICEC'96 contest by differential evolution.*** », *In Proceedings of IEEE International Conference on Evolutionary Computation*. 1996, pp: 842-844.

[51] Thomas V, cour : « ***Optimisation et systèmes dynamiques stochastiques : Recherche heuristique*** », Laboratoire lorrain de recherche en informatique et ses applications, 2011.

[52] Todinca I, « ***Décomposition arborescentes de graphes : calcul, approximations, heuristiques*** », Habilitation à diriger des recherches, Université d'Orléans, Mai 2010.