Nº d'ordre: 02/2020-D/INF

**République Algérienne Démocratique et Populaire**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**
**Université Des Sciences et de la Technologie Houari Boumedienne**
**Faculté d'Électronique et d'Informatique**

**THÈSE DE DOCTORAT EN SCIENCES**
**Présentée pour l'obtention du grade de DOCTEUR**

En: Informatique
Spécialité: Systèmes Intelligents et Ingénierie du logiciel.

PAR
**Mourad LASSOUAOUI**

Thème

# Hyper-heuristiques Intelligentes pour les Problèmes Combinatoires. Application aux problèmes Max-SAT et Classification en Datamining.

Soutenu publiquement le 29/06/2020, devant le jury composé de :

| | | | |
|---|---|---|---|
| **DRIAS Habiba** | Professeur | à l'USTHB | Présidente |
| **BOUGHACI Dalila** | Professeur | à l'USTHB | Directrice de thèse |
| **BENHAMOU Belaid** | Professeur | à l'AMU | Directeur de thèse |
| **LEBAH Yahia** | Professeur | à l'Univ Oran | Examinateur |
| **SI TAYEB Fatima** | Professeur | à l'ESI | Examinatrice |
| **DAOUDI Mourad** | Maitre de conférences/A | à l'USTHB | Examinateur |

# Remerciment

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 à mes directeurs de thèse Prof. Dalila BOUGHACI et Prof. Belaid BENHAMOU pour leur soutien, leur orientation et leurs conseils tout au long de mon parcours. Leur expérience m'ont été d'un apport considérable tout au long de mon travail.
Je remercie vivement Prof. Habiba DRIAS pour l'honneur qu'elle m'a fait en acceptant de présider le jury de ma soutenance de doctorat.
Mes remerciements vont aussi à Prof. Yahia LEBAH, Prof. Fatima SI TAYEB et Dr. Mourad DAOUDI pour l'honneur qu'ils me font d'être dans mon jury de thèse.

# Dédicace

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 mon épouse,
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.

**Résumé**

Cette thèse de doctorat porte sur la résolution de problèmes d'optimisation combinatoire par une approche hyper-heuristique. Nous avons étudié deux fameux problème à savoir le problème Max-SAT et le problème de la sélection d'attributs en datamining. Les deux problèmes sont classés NP-difficiles.

Le problème de la satisfaisabilité booléenne maximale Max-SAT est un problème central en informatique théorique. Il consiste à trouver une assignation des variables propositionnelles qui maximise le nombre de clauses d'une formule de logique propositionnelle. Dans la première partie de notre thèse, nous avons proposé trois approches hyper-heuristiques pour la résolution efficace du problème Max-SAT. Notre première approche combine une stratégie aléatoire de sélection d'heuristiques de bas niveau avec une fonction de choix utilisant un mécanisme d'apprentissage par renforcement (choice function). Dans la deuxième approche, nous avons incorporé le paradigme multiniveau dans la résolution du problème, et ce afin de booster la recherche, notamment sur les grandes instances. Enfin, suite à une étude montrant les limites des stratégies de sélection utilisant un mécanisme d'apprentissage par renforcement additive, nous avons proposé une troisième approche de sélection, que nous avons appelé Synergie Thompson sampling, qui utilise un mécanisme d'apprentissage probabiliste, et qui prend en compte la synergie entre les heuristiques de bas niveau. Toutes les méthodes proposées ont été implémentées et validées sur un ensemble très large d'instances. Les résultats sont très intéressants et montrent l'intérêt de nos approches.

La deuxième partie de notre thèse est consacré au problème de la sélection d'attributs en datamining et classification. Le problème en question consiste à éliminer les attributs redondants ou non pertinents d'un dataset afin d'améliorer la performance. Ceci permettra de réduire la dimension du problème et maximiser la précision de la classification. Pour ce problème, nous avons choisi d'utiliser notre nouvelle approche hyper-heuristique dite «Synergy Thompson sampling». Notre approche a été évaluée sur un ensemble de datasets de répertoire de l'UCI. Les résultats obtenus ainsi que la comparaison avec l'hyper-heuristique la plus récente de l'état de l'art, montrent l'efficacité de notre approche dans la résolution du problème sus-mentionné.

**Mot clés** : Hyper-heuristiques, Max-SAT, sélection d'attributs, Choice function, Synergy Thompson sampling, heuristiques, métaheuristiques.

**Abstract**

This doctoral thesis focuses on solving combinatorial optimization problems using hyper-heuristic approaches. We have studied two well known NP-hard problems: the Max-Sat problem and the feature selection problem in data mining.

The maximum satisfiability problem, or Max-SAT, is of the utmost importance in theoretical computer science. It consists in finding a boolean assignation to the variables that maximizes the number of satisfied clauses of a propositional logic formula. In the first part of our thesis, we propose three hyper-heuristic approaches to solve efficiently Max-SAT. Our first approach combines a random selection strategy of low level heuristics with a choice function that uses a reinforcement learning mechanism. In the second approach, we incorporate the multilevel paradigm to the different hyper-heuristics in order to boost the search process especially on large instances. Finally, by taking into account a recent study showing the limits of additive reinforcement learning techniques, we propose a third selection approach that we named Synergy Thompson sampling. It uses a probabilistic reinforcement learning, and takes into account the synergy between the low level heuristics. All proposed methods have been tested on a large set of instances. The results are very interesting and show the efficiency of our approaches.

The second part of our thesis addresses the feature selection problem in classification. The problem consists in eliminating redundant and irrelevant features from the dataset in order to improve the classifier's performance. This reduces the problem dimensionality and maximizes the accuracy of the classification. For this problem, we use our new approach Synergy Thompson sampling. It has been evaluated on data sets from the UCI repository. The obtained results and the comparison with the most recent hyper-heuristic used in the literature show the efficiency of our approach in this problem.

**Keywords**: Hyper-heuristics, Max-SAT, feature selection, Choice function, Synergy Thompson sampling, heuristics, metaheuristics.

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction Générale

Plusieurs problèmes d'optimisation issus du monde réel sont difficiles à résoudre, et la plupart d'entre eux sont réputés NP-difficiles.

En vue de leur nature exponentielle, les méthodes exactes (complètes), qui se basent sur une recherche exhaustive, n'arrivent pas à résoudre ces problèmes d'une manière efficace. Dans ce cas, une autre catégorie d'algorithmes est utilisée : ce sont les méthodes incomplètes comme les heuristiques et les métaheuristiques.

Plusieurs algorithmes ont montré leurs forces et leurs faiblesses en essayant de résoudre des problèmes d'optimisation combinatoire. Cela a largement contribué au développement de nouvelles techniques d'optimisation. Parmi ces techniques, nous trouvons celles qui font collaborer plusieurs méthodes afin de combler la faiblesse de l'une par la force des autres. Ces méthodes sont appelées **hyper-heuristiques**.

Le terme hyper-heuristique a été introduit par [Cowling et al., 2000]. Ce sont des méthodes approchées avec un haut niveau d'abstraction et qui sont indépendantes du problème traité. Elles gèrent un ensemble de (meta)heuristiques appelées **heuristiques de bas niveau**.

Ayant un ensemble d'heuristiques de bas niveau, une hyper-heuristique de sélection essaie de prédire l'heuristique qui sera la plus appropriée à chaque étape de la recherche.

Plusieurs hyper-heuristiques de sélection sont basées sur l'apprentissage par renforcement. Elles collectent des informations à propos des performances des heuristiques de bas niveau pour apprendre leur comportement, afin de prédire laquelle sera la plus efficace dans la prochaine itération. Un mécanisme d'apprentissage par renforcement est une technique d'apprentissage automatique basée sur les **récompenses** et les **sanctions**. Les hyper-heuristiques les plus usuelles utilisent de simples techniques d'apprentissage par renforcement comme le gradient stochastique (*random gradient*), glouton (*greedy*) ou bien, plus important, l'apprentissage par renforcement additif. Un exemple phare de ce dernier est l'hyper-heuristique ***choice function***.

Une hyper-heuristique utilisant un apprentissage additif affecte un poids à chaque heuristique de bas niveau. Elle augmente le poids de l'heuristique sélectionnée si cette dernière a amélioré la solution courante par rapport à

la fonction "objectif". Elle diminue le poids dans le cas contraire.

Récemment, [Alanazi and Lehre, 2016] ont présenté la première étude théorique qui évalue la performance des mécanismes d'apprentissage par renforcement additifs et montre leurs limites. En effet, cette étude a prouvé théoriquement que si la probabilité de succès d'une heuristique de bas niveau est inférieure à $\frac{1}{2}$, alors cette hyper-heuristique aura le même comportement qu'une hyper-heuristique de sélection aléatoire.

Comme alternative, [Alanazi, 2016] a proposé le mécanisme ***Thompson sampling***. L'hyper-heuristique Thompson sampling utilise un mécanisme d'apprentissage probabiliste, basé sur la loi de probabilité Beta, afin de prédire la meilleure heuristique de bas niveau à exécuter à la prochaine itération. Cependant, cette hyper-heuristique ne prend pas en considération un principe fondamental, qui est la **synergie** entre les heuristiques de bas niveau.

Dans ce projet de doctorat, nous nous sommes intéressés à la résolution de problèmes d'optimisation combinatoire par une approche hyper-heuristique. Notre objectif est de montrer l'efficacité de cette approche pour les problèmes dits NP-difficiles. Nous considérons deux fameux problème dans notre travail : le problème **Max-SAT** et le problème de la **sélection d'attributs** en data-mining.

Max-SAT est un problème central dans plusieurs domaines en informatique comme : l'informatique théorique, l'intelligence artificielle, l'optimisation et la conception et la vérification de matériels.

Le défi actuel en Max-SAT est de résoudre des instances industrielles de très grande taille en un temps raisonnable.

Pour contribuer à la résolution du problème Max-SAT par les approches hyper-heuristiques, nous avons fait appel au mécanisme d'apprentissage par renforcement pour améliorer notre approche. En plus, nous avons incorporé le paradigme **multiniveau pour résoudre efficacement les instances de grande taille**.

Le paradigme multiniveau est basé sur une contraction récursive afin de créer une hiérarchie d'approximations du problème original. Ces approximations sont des échantillons de plus petites tailles et donc plus faciles à résoudre. Au niveau le plus contracté, une solution initiale est calculée, puis elle sera itérativement raffinée et projetée à chaque niveau en partant du niveau le plus contracté jusqu'au problème original.

Dans le cadre de notre projet de thèse, dans un premier temps, nous proposons une approche hybride de sélection d'heuristiques de bas niveau, appelée **stochastique choice function**. Cette dernière combine une méthode de sélection utilisant un mécanisme d'apprentissage par renforcement additif,

appelée choice function, avec un mécanisme de sélection aléatoire. L'approche proposée est évaluée sur des instances de petites, moyennes et grandes tailles du problème Max-SAT. Les résultats obtenus montrent que l'hybridation permet de palier au problème de stagnation constaté dans l'hyper-heuristique choice function.

La deuxième contribution dans notre thèse, est l'incorporation du paradigme multiniveau aux hyper-heuristiques, et ce afin de pouvoir traiter plus efficacement les instances industrielles de très grandes tailles du problème Max-SAT. Des comparaisons avec des méthodes de l'état de l'art ont montré l'apport important de ce paradigme.

Comme troisième contribution, nous proposons une nouvelle hyper-heuristique probabiliste qui prend en considération la synergie entre les heuristiques de bas niveau, appelée **synergie Thompson sampling**. Nous y avons incorporé le paradigme multiniveau. Les résultats obtenus sur les benchmarks industriels du problème Max-SAT confirment l'efficacité de notre approche.

Le deuxième problème que nous avons étudié, dans le cadre de notre thèse, est le problème de la sélection d'attributs. Ce dernier est devenu une étape essentielle en classification, en raison de la complexité et la taille des données à traiter. Le but est d'éliminer les attributs non pertinents, redondants et bruités, afin de n'en garder que ceux permettant d'avoir une meilleure précision lors de la classification. Cette étape permet aussi une meilleure compréhension des résultats et une accélération du temps de traitement des données.

Notre quatrième contribution consiste en la résolution du problème de la sélection d'attributs par une nouvelle approche hyper-heuristique basée sur le concept du "synergie Thompson sampling". La méthode développée est validée sur plusieurs benchmarks du fameux répertoire de l'UCI (*University of California at Irvine*). Aussi, nous avons établi une étude comparative avec l'hyper-heuristique la plus récente trouvée dans l'état de l'art [Montazeri, 2016]. Nos résultats montrent la supériorité de notre approche dans la résolution du problème sus-mentionné.

## Organisation de la thèse

Cette thèse est composée de six chapitres dont nous présentons une brève description comme suit :

- Dans le **chapitre I**, nous présentons les notions de bases liées à l'optimisation combinatoire. Nous donnons aussi un état de l'art sur les méthodes de résolution de problèmes combinatoires notamment les hyper-heuristiques.
- Dans le **chapitre II**, nous exposons le premier problème traité dans cette thèse, à savoir le problème Max-SAT. Nous présentons aussi, le paradigme multiniveau.

- Dans le **chapitre III**, nous abordons le deuxième problème étudié dans cette thèse, à savoir le problème de sélection d'attributs dans la classification.

- Dans le **chapitre IV**, nous détaillons nos deux premières contributions qui sont : L'hyper-heuristique stochastique choice function, et l'intégration du paradigme multiniveau aux hyper-heuristiques. Les expérimentations sont faites sur des instances de petites, moyennes et grandes tailles du problème Max-SAT. Une comparaison avec les méthodes GSAT et Walk-SAT est également présentée.

- Le **chapitre V** est consacré à notre troisième contribution qui est l'hyper-heuristique synergie Thompson sampling. Dans ce chapitre, notre nouvelle approche est validée sur des instances industrielles de très grandes tailles du problème Max-SAT.

- Et enfin, le **chapitre VI** est dédié à la quatrième contribution qui concerne l'application de notre nouvelle approche (synergie Thompson sampling) sur le problème de la sélection d'attributs dans la classification. Les résultats sur les benchmarks UCI sont exposés, ainsi qu'une comparaison avec l'état de l'art.

- Enfin, nous terminons notre document par une conclusion générale et quelques perspectives de notre travail.

# Chapitre 1

# Optimisation, métaheuristiques et hyper-heuristiques

## 1.1 Introduction

Les problèmes d'optimisation combinatoire sont d'une importance capitale dans plusieurs domaines, tel que l'informatique, les mathématiques, la recherche opérationnelle, l'industrie,... etc. l'obtention d'une solution de bonne qualité en un temps raisonnable est déterminant, surtout, que de nos jours, nous somme confronté à des tailles de données de plus en plus grandes. Dans ce chapitre, nous parlons des différentes techniques de résolution de ce genre de problème, qui sont les méthodes exactes, les méthodes heuristiques, les métaheuristiques et nous terminons avec les hyper-heuristiques.

## 1.2 Théorie de la complexité et classes de problèmes

La théorie de la complexité [Papadimitriou, 1994] s'intéresse à l'étude formelle de la difficulté des problèmes en informatique. Afin de comparer les algorithmes qui résolvent le même problème entre eux, une démarche indépendante des caractéristiques de la machine ou des langages de programmation est nécessaire. En utilisant les outils mathématiques, nous pouvons calculer une mesure de la complexité d'un algorithme, en nombre d'opérations élémentaires nécessaires pour que l'algorithme fournisse la solution du problème traité.

Les travaux de Stephen Cook [Cook, 1971] et Richard Karp [Karp, 1972], entre autres, montrent qu'il existe une classification des problèmes complexes. On distingue alors deux classes principales :

- La classe P : des problèmes qui sont calculables en temps polynomial par une machine de Turing déterministe. Une machine de Turing déterministe a au plus une transition possible, contrairement à une machine de Turing non déterministe, qui peut en avoir plusieurs.

- La classe NP (*Non-deterministic Polynomial time*) des problèmes qui sont calculables par une machine de Turing non déterministe en temps

polynomial (c'est-à-dire qu'on peut tester la validité d'une solution du problème en un temps polynomial).

Une question centrale qui reste soulevée est de savoir si P $\stackrel{?}{=}$ NP.

Les problèmes NP-complets sont une sous classe des problèmes NP qui sont plus difficiles à résoudre. Aucun d'entre eux n'a pu être résolu, à ce jour, par un algorithme polynomial, malgré les efforts déployés. La particularité de cette classe est que si l'on trouvait un algorithme polynomial permettant de résoudre un seul problème NP-complet, on pourrait en déduire un autre pour chacun des autres problèmes de la classe NP.

Les problèmes NP-difficiles sont au moins aussi difficiles que les problèmes NP-complets, généralement leurs réponse est de type numérique. De nombreux problèmes d'optimisation combinatoire ont été prouvés NP-difficiles. Cette difficulté n'est pas seulement théorique, mais se confirme aussi dans la pratique.

## 1.3 Méthodes exacte Vs Méthodes approchées

Pour résoudre les problèmes NP-difficile ; il existe deux types de méthodes de résolution : les méthodes exactes et les méthodes approchées.

Les méthodes exacte – complète – garantissent l'obtention de la solution optimale du problème traité, elles se basent généralement sur une recherche arborescente, ces méthodes explorent de manière exhaustive l'espace de recherche en évaluant les solutions admissibles, et à la fin du processus elles retournent la meilleure solution du problème, ce qui provoque une explosion combinatoire pour des instances de problème de grandes tailles, ce qui les rends inutilisables dans ces cas-là.

A l'inverse des méthodes exactes, les méthodes approchées ne garantissent pas d'obtenir la solution optimal du problème traité, elles fournissent généralement une « bonne solution » (satisfaisante) en un temps « raisonnable », c'est justement pour cette dernière que les heuristiques sont souvent utilisées, car le problème que nous rencontrons généralement dans les méthodes exactes est l'explosion combinatoire.

## 1.4 Les méthodes heuristiques

L'origine du mot heuristique vient du mot grec *heuriskein*, qui signifie « je trouve » ou elle est souvent vue comme étant « l'art d'inventer » ou « l'art de découvrir »[1]. Beaucoup la définissent comme étant un ensemble d'outils, de techniques, de procédés permettant la découverte ou l'invention [Meignan, 2008]. Donc en résumé, les heuristiques exploitent les informations du domaine traité, afin de guider le processus de recherche.

Les heuristiques sont utilisées dans plusieurs méthodes de résolution (exactes ou approchées). Elles permettent à ces dernières d'améliorer leurs performances en temps de calcule et en qualité de la solution trouvée.

[1] https ://fr.wikipedia.org/wiki/Heuristique_(mathématiques)

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 d'assez bonne qualité avec une exécution plus rapide que les méthodes exhaustives pour des instances de problèmes de grandes tailles [Dréo, 2004].

Comme exemple, nous pouvons citer le problème du voyageur de commerce, qui consiste à trouver le plus court chemin qui passe par toutes les villes d'un graphe, où chaque ville est visitée une et une seule fois. Il consiste donc à trouver le plus court cycle Hamiltonian dans un graphe, ce qui le rend NP-difficile. Parmi les heuristiques utilisées pour ce problème nous pouvons citer [Nilsson, 2003] :

- **Le plus proche voisin** : qui sélectionne toujours la plus proche ville à visiter comme suit :
  1. Sélectionner une ville aléatoirement.
  2. Trouver la plus proche ville non visitée et la sélectionner.
  3. S'il reste des villes non visitées alors répéter l'étape 2.
  4. Retourner à la ville de départ.
- **Glouton** : construit le tour en sélectionnant toujours l'arête la plus courte comme suit :
  1. Trier toutes les arêtes.
  2. Sélectionner l'arête la plus courte et la rajouter au tour si elle respecte les contraintes du problème.
  3. Si nous n'avons pas encore les N arêtes alors répéter l'étape 2.

## 1.5 Les métaheuristiques

### 1.5.1 Introduction

Les métaheuristiques sont des méthodes de recherches approchées qui se veulent génériques et complétement indépendantes des problèmes à résoudre. Ces méthodes ont changé la façon de résoudre des problèmes difficiles en s'inspirant parfois de phénomènes naturels [Gardeux, 2011].

Le terme métaheuristique a été utilisé pour la première fois par Glover en 1986, après avoir présenté la recherche taboue [Glover, 1986]. Le mot métaheuristique est le composé de deux termes grec : le suffixe méta (au-delà) et le verbe *heuriskein* (trouver).

Les métaheuristiques parcourent l'espace de recherche du problème afin de trouver son **optimum global**, tout en évitant les **optimums locaux** (Figure 1.1). Ceci peut se faire grâce à un bon paramétrage (s'il existe), des opérations de génération de solution efficace,... etc.

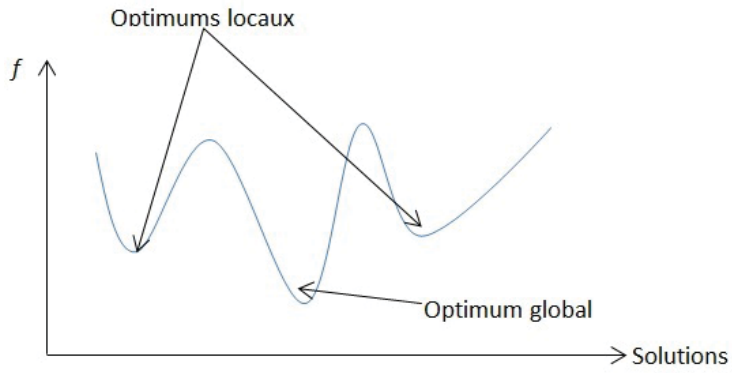



FIGURE 1.1 – Exemple d'optimum local et global pour un problème de minimisation.

La phase de paramétrage permet à l'algorithme de mieux parcourir l'espace de recherche. Cependant, pour arriver à un bon paramétrage, il faut tester différentes combinaisons de valeurs ce qui peut être fastidieux.

La condition d'arrêt dans les métaheuristiques, contrairement aux méthodes exactes qui s'arrêtent après avoir trouvé la solution optimale, est fixée par rapport au nombre d'itérations de l'algorithme, le temps d'exécution,... etc.

### 1.5.2 Concepts de Diversification et Intensification

En parcourant l'espace de solutions d'un problème, les métaheuristiques risquent d'être bloquées dans des régions d'optimums locaux. Elles risquent aussi de rater l'optimum global, si elles n'explorent pas assez la zone. C'est la raison pour laquelle, une bonne métaheuristique se doit de faire un équilibre entre deux concepts : la **diversification** et l'**intensification**.

La diversification (Figure 1.2 (a)) permet aux métaheuristiques de s'échapper des optimums locaux, et d'explorer d'autres zones de l'espace de recherche, afin de tomber sur celle pouvant contenir l'optimum global. L'intensification (Figure 1.2 (b)), comme son nom l'indique, intensifie la recherche dans la zone en cours afin d'améliorer le plus possible la solution courante.

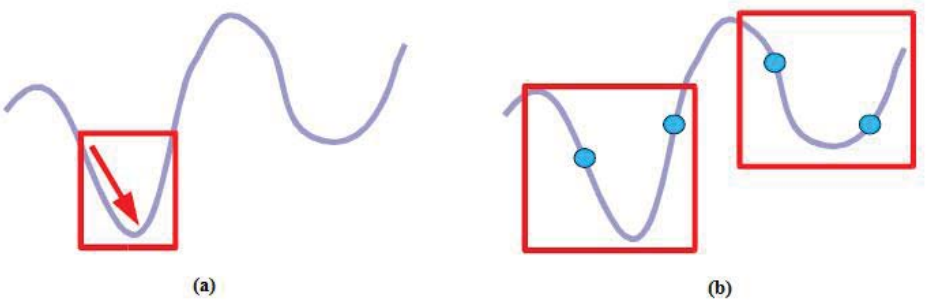



FIGURE 1.2 – Principe d'intensification (a), et de diversification (b) [Dréo, 2004].

L'équilibre entre ces deux concepts est donc très important. En effet, si la métaheuristique penche plus vers l'intensification alors elle risque d'être

bloquée dans un optimum local. Par contre, si elle se penche plutôt vers la diversification, alors elle risque de parcourir plusieurs zones de l'espace de recherche sans pour autant atteindre l'optimum global.

### 1.5.3 Classification des métaheuristiques

Dans la littérature, plusieurs classifications des métaheuristiques existent [Sirenko, 2009]. Cette diversité est le fruit de différents points de vue, en prenant en considération plusieurs caractéristiques en commun.

Dans [Talbi, 2009] plusieurs caractéristiques sont décrites :

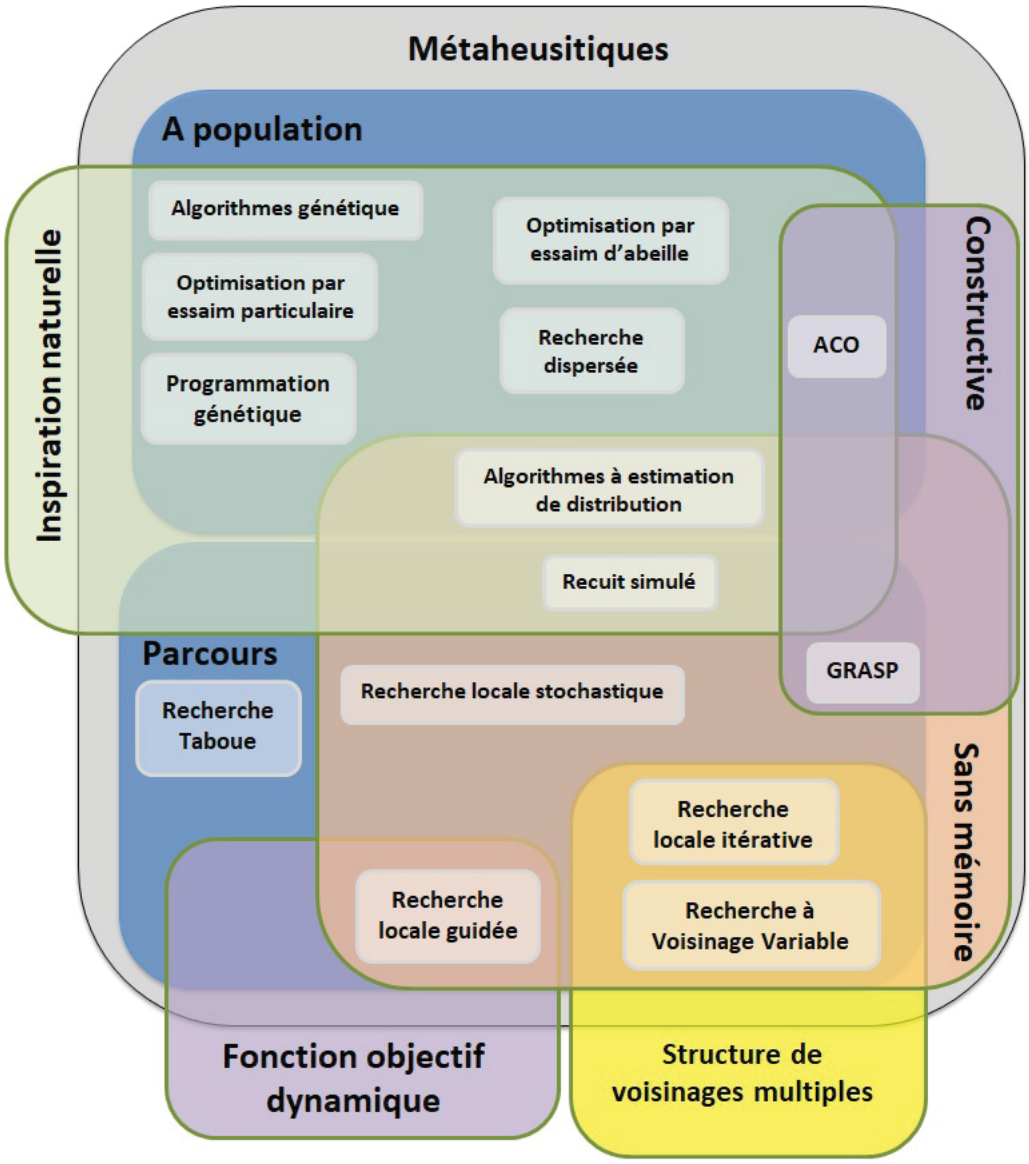



FIGURE 1.3 – Classification des métaheuristiques sous plusieurs angles.

- **Inspiré de la nature ou non** :

  Plusieurs algorithmes de métaheuristique s'inspirent de la nature. Cela peut être un phénomène physique, comme le recuit simulé (SA), où l'al-

gorithme s'inspire du processus de recuit en métallurgie, qui tente de modifier le solide pour le rendre de meilleur qualité et ceci en cherchant un état d'énergie stable. Comme on peut avoir des métaheuristiques inspirées d'insectes comme les Fourmies (ACO), où l'algorithme reproduit le même comportement de ces êtres cherchant la nourriture, et ce en suivant les phéromones secrétées par celles-ci.

Parmi les métaheuristiques qui ne sont pas inspirées de la nature, nous pouvons citer la recherche locale itérative (ILS), ou la recherche locale stochastique (SLS).

- **Basée population de solution ou une seule solution** :

  Un autre critère qui peut être employé lors de la classification, est le nombre de solutions utilisées en même temps. Les métaheuristiques basées population, décrivent l'évolution d'un ensemble de points (de solutions) dans l'espace de recherche, cette ensemble est appelé **population**. Nous pouvons citer comme exemple les algorithmes génétiques (GA), où cet algorithme gère un ensemble d'**individus** et effectue sur cet ensemble trois opérations qui sont : la sélection, la mutation et le croisement. Tant dis que pour les métaheuristiques basées solution unique, ou appelées les **métaheuristiques de parcours**, elles ne font évoluer qu'une seule solution à chaque itération, comme la recherche taboue (TS) et le recuit simulé,... etc.

- **Fonction objectif statique ou dynamique** :

  Pour évaluer la qualité d'une solution, les métaheuristiques utilisent des fonctions objectif. La fonction objectif peut être **statique**, c'est-à-dire elle ne change pas sa formule. Les métaheuristiques citées précédemment font toutes partis de cette classe.

  Par contre, une autre classe existe, celle des méthodes utilisant des fonctions objectif **dynamiques**. Elles font évoluer cette fonction objectif en la modifiant, ce qui change le paysage de l'espace de recherche afin de sortir des optimums locaux. L'une des métaheuristiques utilisant une fonction objectif dynamique est la recherche locale guidée (GLS).

- **Avec mémoire ou sans mémoire** :

  Un autre critère de classification est l'utilisation de l'historique de recherche (emploi de la mémoire) afin d'influencer le résultat des prochaines itérations. Les algorithmes sans mémoire effectuent un processus de Markov, car ils utilisent l'état actuel (la solution courante) pour déterminer la prochaine action, nous pouvons citer le recuit simulé ou la recherche locale stochastique.

  Pour les métaheuristiques utilisant la mémoire, elles sauvegardent les mouvements récemment effectués, les solutions visitées ou les décisions prises. La mémoire, donc, permet d'éviter de « revisiter » les solutions récentes et ainsi éviter les cycles, un exemple la recherche taboue.

- **Un ou plusieurs voisinages** :

  La plupart des métaheuristiques travaillent sur une structure unique de voisinage. D'autre, tel que la recherche à voisinage variable (VNS), utilisent un ensemble de structures de voisinages qui produisent une meilleure diversification.

- **Constructive ou perturbatrice** : L'autre critère de classification des métaheuristiques est la nature du processus de recherche, c'est-à-dire, comment la métaheuristique manipule la solution.

  La première classe englobe les métaheuristiques **constructives**. Elles démarrent à partir d'une solution vide et construisent petit à petit une solution complète.

  Pour la deuxième classe, les métaheuristiques **perturbatrices** démarrent à partir d'une solution initiale complète et tentent à chaque étape de l'améliorer.

La figure 1.3 résume les différents critères de classification des métaheuristiques avec quelques exemples pour chaque cas.

Dans ce domaine de l'optimisation d'autres modèles peuvent apparaitre, et ceci grâce au développement d'autres disciplines (médecine, mécanique,. . . etc).

## 1.6 Les hyper-heuristiques

### 1.6.1 Introduction

Il existe d'autres méthodes approchées de plus haut niveau, indépendantes du problème [Kendall et al., 2002], faisant coopérer plusieurs métaheuristiques et/ou algorithmes heuristiques spécifiques au problème à traiter, ces méthodes ont pour nom **Hyper-heuristiques**.

Le terme hyper-heuristique a été employé en 2000 [Burke et al., 2019], et décrivait 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)[Kendall and Mohamad, 2004], le problème de l'emploi du temps des examens (Examination Timetabling Problem) [Özcan et al., 2009, Burke et al., 2012, Sin, 2011, Kendall and Hussin, 2004, Özcan et al., 2012], problème de planification [Cowling et al., 2000], le problème du gagnant dans les enchères combinatoires [Lassouaoui and Boughaci, 2014] etc.

Ce principe n'est pas nouveau, puisqu'il a vu le jour dans les années 60, où Fisher et Thompson [Fisher and Thompson, 1961, Fisher and Thompson, 1963] ont conclus, que dans le problème d'ordonnancement, il est plus efficace de combiner plusieurs règles que d'utiliser chaque règles séparément.

Les hyper-heuristiques n'effectuent pas directement la recherche sur l'espace de solutions, mais indirectement à travers les méthodes de résolution du problème (métaheuristiques et/ou algorithmes de recherche heuristique), et donc travaille sur l'espace de méthodes [Burke et al., 2013, Soubeiga, 2003,

Rodríguez et al., 2007a, Özcan et al., 2009] comme illustré dans la figure 1.4. Ce qui nous conduit à dire que les hyper-heuristiques tentent de trouver, à partir d'un ensemble d'(méta)heuristiques, la bonne méthode (la plus appropriée) dans une situation particulière, au lieu d'essayer de résoudre le problème directement [Burke et al., 2010].

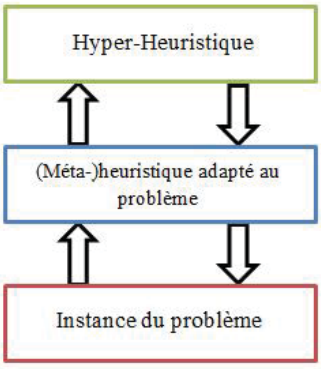



FIGURE 1.4 – Différents niveaux d'interaction dans une plateforme hyper-heuristique.

Certaines hyper-heuristiques exploitent des informations indépendantes du problème afin de décider quelle (méta)heuristique choisir. Ces informations sont des indicateurs de performance, par exemple : temps CPU de chaque exécution, la qualité d'une solution trouvée, etc. [Soubeiga, 2003, Cowling et al., 2000]. Comme il existe certaines hyper-heuristiques faisant leurs sélection en utilisant une stratégie inspirée de la nature : Algorithme Génétique [Rodríguez et al., 2007b, Rodríguez et al., 2007a, Ochoa et al., 2009, Garrido and Riff, 2007a, Garrido and Riff, 2007b, Pillay, 2008], le recuit simulé [Dowsland et al., 2007, Bai et al., 2012],... etc.

Comme les (méta)heuristiques possèdent différentes forces et faiblesses, les hyper-heuristiques tentent de combiner ces méthodes afin d'arriver à compenser les faiblesses de certaines par la force des autres [Soubeiga, 2003].

Une nouvelle définition des hyper-heuristiques est apparue, après que la programmation génétique fut utilisée comme hyper-heuristique [Burke et al., 2010]. 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* ».

La figure 1.5 montre l'architecture des hyper-heuristiques de sélection et celle des hyper-heuristiques de génération.

### 1.6.2 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,
- Des composants ou sous-programme.

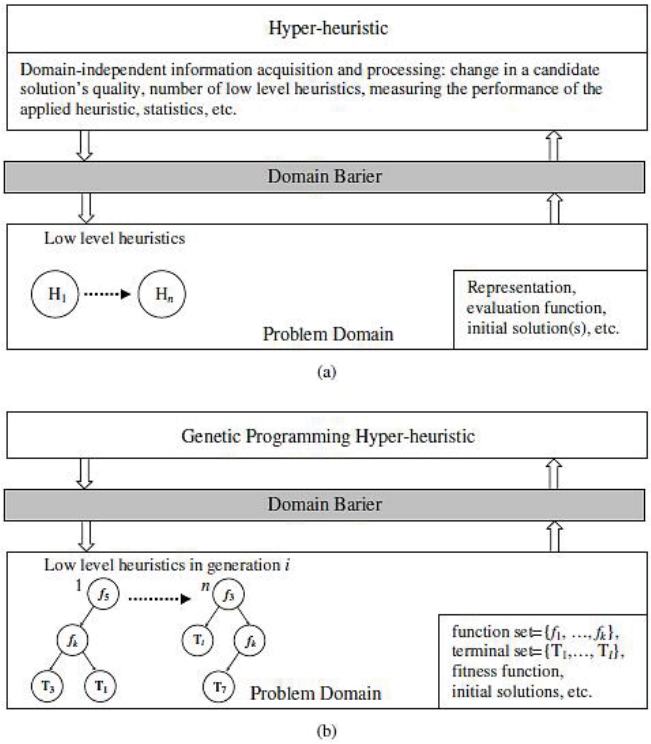



FIGURE 1.5 – Hyper-heuristique de sélection (a) vs Hyper-heuristique de génération (b) [Burke et al., 2009]

Cet ensemble est appelé ensemble d'**heuristiques de bas niveau**, dans le but de produire des solutions résolvant de manière efficace ces problèmes.

Les auteurs de [Burke et al., 2010] présentent les différentes classifications faites sur les hyper-heuristiques, à commencer par [Soubeiga, 2003] 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 [Bai, 2005] où l'auteur considère qu'il existe deux classes : (i) les hyper-heuristiques manipulant des heuristiques de bas niveau constructives, ou (ii) les hyper-heuristiques travaillant avec des méthodes perturbatrices. 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'améliorer.

[Chakhlevitch and Cowling, 2008] présentent 4 classes d'hyper-heuristiques : (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 [Burke et al., 2010], les auteurs se basent sur les précédentes classifications afin de présenter 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 techniques d'apprentissage.

La figure 1.6 illustre la classification des hyper-heuristiques proposée par

[Burke et al., 2013]. Pour la première dimension nous avons :

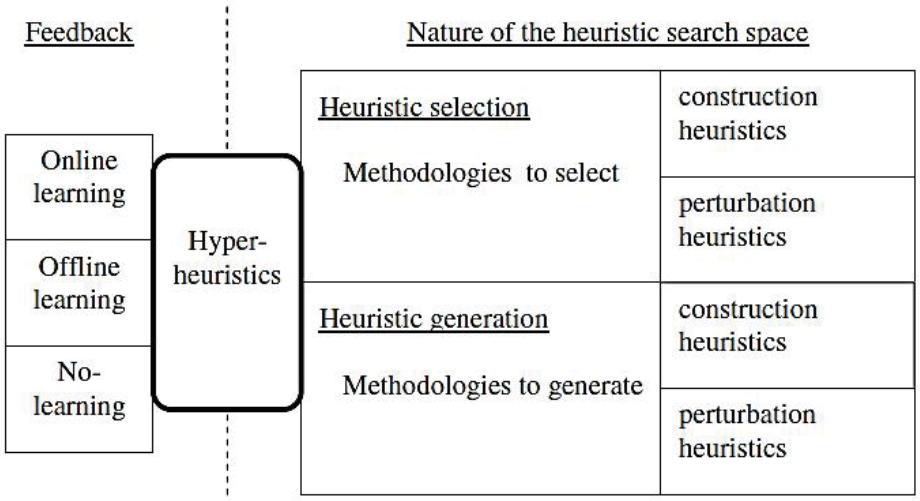



FIGURE 1.6 – Classification des hyper-heuristiques selon [Burke et al., 2013]

- **Module de sélection d'heuristiques** (*heuristic selection*) : qui est une méthode qui choisit une heuristique de bas niveau à partir d'une base.
- **Module de génération d'heuristiques** (*heuristic generation*) [Burke et al., 2009] : 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.

[Burke et al., 2010, Burke et al., 2013, Özcan et al., 2009] décomposent l'hyper-heuristique perturbatrice en deux composantes principales :

- **Module de sélection** : qui effectue la tache de sélection d'heuristiques 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 renvoyée par une heuristique de bas niveau.

Dans ce qui suit, nous allons nous intéresser principalement aux hyper-heuristiques de sélection.

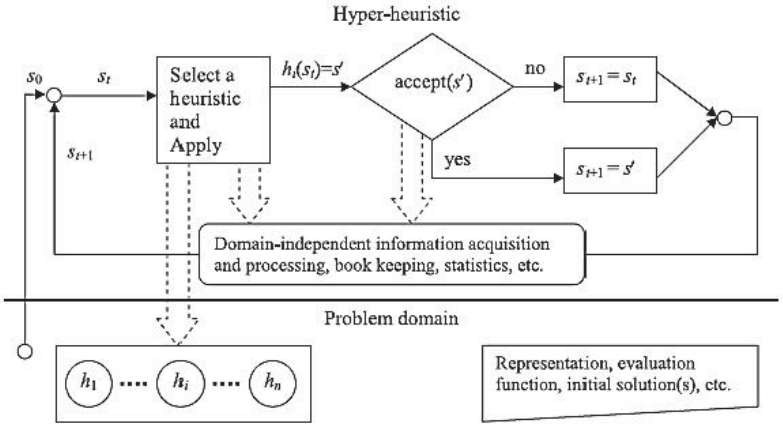



FIGURE 1.7 – Hyper-heuristique de sélection d'heuristiques de bas niveau perturbatrices [Burke et al., 2012]

### 1.6.3 Les hyper-heuristiques de sélection

Une hyper-heuristique de sélection choisis l'heuristique de bas niveau la plus appropriée, au cours du processus de recherche à partir d'un ensemble prédéfini, comme illustré dans la figure 1.7.

Diverse stratégies de sélection ont vu le jour, certaines d'entre elles s'inspirent des métaheuristiques. Ces méthodes agissent indirectement sur le problème en manipulant les heuristiques de bas niveau suivant leurs natures. Nous pouvons citer comme exemple, l'hyper-heuristique taboue, où cette hyper-heuristique met dans sa mémoire taboue non pas une solution, mais une heuristique de bas niveau selon sa stratégie (tel que : quel que soit son résultat, si l'heuristique de bas niveau n'a pas améliorer, . . . ). Cette hyper-heuristique a été utilisée dans plusieurs travaux [Burke et al., 2003, Kendall and Hussin, 2004, Dowsland et al., 2007, Zamli et al., 2016]. Une autre hyper-heuristique inspirée d'une métaheuristique souvent employée est l'algorithme génétique [Rodríguez et al., 2007b, Rodríguez et al., 2007a, Ochoa et al., 2009, Garrido and Riff, 2007a, Garrido and Riff, 2007b, Pillay, 2008]. Dans cette hyper-heuristique, un chromosome ne représente pas une solution du problème, mais une séquence d'heuristiques de bas niveau à exécuter. Cette séquence sera évaluée, et les opérations de mutation et de croisement seront appliquées sur cet ensemble de chromosomes.

Il existe aussi des hyper-heuristiques utilisant des mécanismes d'apprentissage. Nous pouvons citer l'hyper-heuristique *choice function* [Cowling et al., 2000] qui a été utilisée dans plusieurs problèmes tel que le problème de la détermination du gagnant dans les enchères combinatoires [Lassouaoui and Boughaci, 2014], le problème de conception de la résistance au choc du véhicule [Maashi et al., 2015], pour le problème de planification de projets [Cowling et al., 2002],. . . etc. Choice function est une technique utilisant un mécanisme d'apprentissage par renforcement additif, qui attribue à chaque heuristique de bas niveau un poids. En effet, cette technique permet de juger l'efficacité d'une heuristique de bas niveau afin de décider laquelle sera exécutée à la prochaine itération. 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 a été appelée pour la dernière fois.

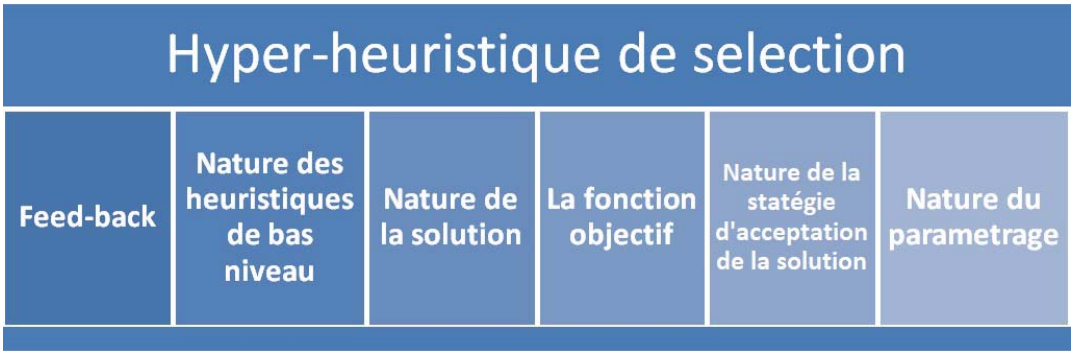



FIGURE 1.8 – Classes des hyper-heuristiques de sélections

Dans [Burke et al., 2019], les auteurs décrivent plus en détails les différentes classes que peuvent former les hyper-heuristiques de sélection comme indiqué dans la figure 1.8. En plus de la nature de l'apprentissage (*feed-back*) et de la nature des heuristiques de bas niveau (constructives ou perturbatrices), présentés précédemment, les auteurs ont ajouté d'autres points de différenciations :

- **Nature de la solution** :

  Les hyper-heuristiques basées population (*multi-point*) manipulent plusieurs solutions courantes durant le processus de recherche. Cependant, les hyper-heuristiques basées solution unique (*single-point*) manipulent une seule solution courante. La majorité des hyper-heuristiques de sélection sont basées solution-unique, pouvant avoir aussi quelques heuristiques de bas niveau basée population. Il existe aussi des hyper-heuristiques utilisant une approche qui combine le *single-point* et *multi-point* durant le processus de recherche selon des phases.[Hsiao et al., 2012, Lehrbaum and Musliu, 2012]

- **La fonction objectif** :

  Une hyper-heuristique, comme c'est le cas des métaheuristiques, peut avoir une fonction mono-objectif ou bien multi-objectif selon le problème à résoudre.

- **La nature de la stratégie d'acceptation de la solution** :

  Les mécanismes d'acceptation de solution d'une hyper-heuristique de sélection, peuvent être classifiés en deux catégories :

  – Stochastique : si les probabilités sont utilisées dans le processus de décision (d'accepter ou de rejeter une solution), exemple : le recuit simulé.

  – Non stochastique : Cette catégorie contient deux sous-catégorie :

  - Méthode basique : par exemple : **Toute les solutions sont acceptés** (*All moves*), ou **Accepter que les solutions améliorante** (*Only Improvement*)
  - Méthode de seuil : par exemple ***Great Deluge***, où cette méthode calcule un seuil de tolérance selon lequel elle accepte une solution ou pas.

- **Nature du paramétrage** :

  Le module de sélection des heuristiques de bas niveau, le module d'acceptation et les heuristiques de bas niveau peuvent avoir des paramètres qui permettent de les contrôler. Ces paramètres peuvent être **statiques**, donc, doivent être fixés avant le début du processus de recherche, ou bien, **dynamiques** donc elles changent de valeurs selon une manière prédéfinie au cours du processus de recherche.

## 1.7 Conclusion

Les méthodes approchées sont indispensables pour aborder des problèmes de grande taille. Les méthodes heuristiques et métaheuristiques ont été développées dans ce sens. Cependant, chacune d'elles possèdent des avantages et des inconvénients. Des méthodes de plus haut rang, appelées hyper-heuristiques, ont été développées pour faire collaborer ces différentes méthodes afin de compenser les faiblesses des uns par la force des autres.

Dans notre travail, nous nous intéressons aux hyper-heuristiques et à leurs application sur deux problèmes combinatoires très connue, à savoir le problème Max-SAT et le problème de sélection d'attributs dans la classification. Dans le prochain chapitre, nous abordons le problème Max-SAT et le paradigme multiniveau qui permet de traiter des instances de très grandes tailles.

# Chapitre 2

# Problème Max-SAT et le paradigme multiniveau

## 2.1 Introduction

Dans ce chapitre, nous parlons d'un problème auquel nous nous intéressons dans cette thèse qui est le problème Max-SAT. Nous abordons également, un paradigme qui permet d'améliorer les performances des méthodes de résolution sur des instances de grandes tailles, qui est le paradigme multiniveau.

Nous commençons par une définition du problème SAT et de sa variante Max-SAT, puis, nous citons quelques domaines d'application de ce problème afin de souligner l'importance de ce dernier. Ensuite, sans être exhaustif, nous exposons les différentes méthodes de résolution de l'état de l'art. Nous passons ensuite vers le paradigme multiniveau. Après la définition, nous exposons son principe général et les conditions de son applicabilité. Nous terminons, en citons des travaux de l'état de l'art utilisant ce paradigme.

## 2.2 Le problème de satisfiabilité (SAT)

### 2.2.1 Définition

SAT est le premier problème prouvé NP-complet [Cook, 1971]. Il concerne la satisfiabilité d'une formule logique. Généralement, une instance SAT est définie par une formule logique $\Phi$ représentée sous une forme normale conjonctive (CNF) comme suit [Cook, 1971] :

$$\Phi = \wedge_{j=1}^{m} C_j$$

$$C_j = (\bigvee_{k \in I_j} x_k) \vee (\bigvee_{k \in \bar{I}_j} \bar{x}_k)$$

Où $C_j$ représente une clause qui est une disjonction de littéraux, un littéral étant une variable logique ou sa négation, $m$ est le nombre de clauses et $n$ est le nombre de littéraux, $x_i$ est un littéral et $\bar{x}_i$ est sa négation. $\bar{I}_j$ est un sous ensemble de n littéraux et $I_j$ est un sous ensemble de la négation de $n$ littéraux $I_j \cap \bar{I}_j = 0$.

Résoudre une instance SAT revient à trouver une assignation de valeurs booléennes (VRAI, FAUX) aux variables afin que la formule soit évaluée à VRAI.

La manière de résolution la plus intuitive serait de tester toutes les solutions. Cela représente $2^n$ possibilités, $n$ étant le nombre de variables.

Le problème SAT possède plusieurs variantes [Biere et al., 2009] tel que :

- Le problème **k-SAT** est dérivé du problème SAT en imposant des clauses de k variables. Un intérêt particulier est porté sur 2-SAT et 3-SAT. En effet, 2-SAT peut être résolu en un temps linéaire et k = 3 est la plus petite valeur pour laquelle k-SAT est NP-complet.

- **Horn-SAT** est représenté par une formule où chaque clause a au plus une variable sans négation, il peut être résolu en un temps linéaire.

Dans cette thèse, nous nous intéressant à la variante **Max-SAT**, qui est une variante d'optimisation du problème classique SAT. Résoudre un problème Max-SAT revient à chercher une assignation aux variables afin de satisfaire le plus grand nombre de clauses. Max-SAT possède d'autres variantes qui sont :

- **Max-W-SAT**, où on associe à chaque clause un poids. L'objectif étant de trouver une assignation aux variables qui maximise la somme des poids des clauses satisfaites simultanément.

- **Partial MAX-SAT** est une hybridation des problèmes SAT et MAX-SAT. Une instance Partial Max-SAT est constituée de deux ensembles de clauses D1 et D2, où il faut satisfaire toutes les clauses de D1 et le maximum de clauses de D2.

Le problème Max-SAT a été utilisé dans plusieurs domaines tel que : les bases de donnée [Miyazaki, 1996], l'électronique pour le problème FPGA (*Field Programmable Gate Array*) [Li, 2004], le problème d'ordonnancement [Cha et al., 1997], le problème de routage [Xu et al., 2003], le problèmes de vérification de modèles pour les systèmes à états finis (*model checking*) [Biere et al., 1999], le problèmes de planification [Rintanen et al., 2006], le domaine des marchés virtuels [Sandholm, 2002] et le domaine d'intégration à grande échelle (VLSI) [Smith et al., 2005].

## 2.3 Méthodes de résolution pour Max-SAT

### 2.3.1 Méthodes exactes

Les méthodes exactes se basent sur une recherche exhaustive, afin de trouver la solution exacte ou pour vérifier la satisfiabilité d'une instance SAT. Puisque le nombre de solutions possibles est $2^n$, les méthodes exactes ont une complexité exponentielle [Menaï and Batouche, 2005], elles se basent en générale sur l'exploration de l'arbre de recherche en largeur d'abord (*Breadth First Search* ou **BFS**) ou bien en profondeur d'abord (*Depth First Search* ou **DFS**).

La plus part des solveurs Max-Sat de l'état de l'art se basent sur l'algorithme DPLL [Davis et al., 1962] (Davis–Putnam–Loveland) comme le ***branch and bound*** [Li et al., 2005], notamment le solver Maxsatz [Li et al., 2009]. Le principe général du *branch and bound* est comme suit :

- L'espace de toutes les solutions possibles d'une formule CNF $\Phi$, peut être représenté sous la forme d'un arbre de recherche, où les nœuds représentent des affectations partielles et les feuilles représentent des affectations complètes. Un algorithme branch and bound pour Max-SAT, explore l'arbre de recherche en profondeur d'abord. A chaque nœud, l'algorithme compare deux nombres :
  - Le nombre de clauses insatisfaites obtenu à partir de la meilleure solution trouvée jusqu'alors, appelé **borne supérieure** ou (*upper bound* (UB)).
  - Le nombre de clauses insatisfaites obtenu avec la solution courante (*unsat*), additionné au nombre sous-estimé du nombre de clauses qui deviendront insatisfait si on étend la solution partielle courante à une solution complète (underestimation). La somme $unsat + underestimation$ est appelée **borne inférieure** (*lower bound* (LB)).
- Si, à un moment de la recherche, $UB \leq LB$ alors on ne peut plus trouver de meilleure solution. Dans ce cas, l'algorithme élague le sous arbre et reviens en arrière (*backtraking*) vers un nœud plus haut dans l'arbre de recherche.
- Si $UB > LB$, l'algorithme étend la solution courante en rajoutant une variable, ce qui forme deux branches : celle de gauche correspond à une affectation de la valeur FAUX, et celle de droite à une affectation de la valeur VRAI. Dans ce cas, la formule associée à la branche de gauche (resp. droite) est obtenue à partir de la formule du nœud courant en supprimant toutes les clauses contenant le littéral $\neg p$ (resp. $p$) et en supprimant toute les occurrences du littéral $p$ (resp. $\neg p$) ; c-a-d, L'algorithme applique la règle du littéral unique (*one-literal rule*).
- La solution de l'instance Max-SAT est la valeur que prend UB après l'exploration de tout l'arbre.

L'algorithme 1 montre en pseudo code le processus basique d'un algorithme branch and bound pour le problem Max-SAT.

L'algorithme utilise les notations suivantes :

- **empty-clauses**($\Phi$) est une fonction qui retourne le nombre de clauses vides de $\Phi$.
- $LB(\Phi)$ est la borne inférieure (*lower bound*) de $\Phi$.
- $UB$ est la borne supérieure (*upper bound*) du nombre de clauses insatisfaites obtenues avec une solution optimale. Sa valeur initiale est $\infty$.

**Algorithm 1** Branch and Bound

**Entrée(s)** max-sat(Φ,UB) : Φ : formule CNF ; UB : borne supérieure.
**Sortie(s)** le nombre minimum des clauses insatisfaites de Φ.

```
1: Si (Φ = ∅ OU Φ contient seulement des clauses vides) Alors
2:    Retourner empty-clauses(Φ)
3: Finsi
4: Si (LB(Φ) ≥UB) Alors
5:    retourner ∞.
6: Finsi
7: p :=select-variable(Φ)
8: UB :=min(UB,max-sat(Φ_¬p,UB))
9: Retourner min(UB,max-sat(Φ_p,UB))
```

- **select-variable**(Φ) est une fonction qui retourne une variable de Φ suivant une heuristique.
- $\Phi_p(\Phi_{\neg p})$ est la formule obtenue en appliquant la règle du littéral unique (*one-literal rule*) à Φ en utilisant le littéral $p(\neg p)$.

Les méthodes exactes se basant sur cet algorithme, implémentent en plus des techniques de pré-traitements, des heuristiques intelligentes pour la sélection de variables, des techniques d'inférences puissantes, des bornes inférieures de bonne qualité et des structures de données efficaces. Parmi les plus recente nous citons [Argelich et al., 2018] qui proposent de nouvelles règles d'inférence pour Max-SAT.

Il existe aussi un large éventail de méthodes exactes qui se basent sur les règles d'inférence du problème Max-SAT et utilisent des heuristiques et des méthodes d'apprentissage afin d'accélérer la recherche [Heras et al., 2007, Lin and Su, 2007, Larrosa et al., 2008, Ramírez and Geffner, 2007, Alsinet et al., 2004, Argelich and Manya, 2007].

### 2.3.2 Méthodes approchées

Selon [Liu and De Melo, 2017], les méthodes approchées se basant sur des recherches locales, et ayant une fonction objectif qui cherche à minimiser le nombre de clauses insatisfaites, obtiennent des résultats corrects lors de la résolution du problème SAT et le problème Max-SAT. Comme par exemple les méthodes GSAT et WalkSAT.

L'un des premiers algorithmes de recherche locale pour résoudre SAT est **GSAT** [Selman et al., 1992]. GSAT commence par générer une affectation aléatoire aux variables, puis utilise l'heuristique de la plus grande pente, afin de trouver la nouvelle affectation qui augmente le plus le nombre de clauses satisfaites. Après un nombre fixe d'itérations, la recherche est relancée à partir d'une nouvelle affectation aléatoire. La recherche se poursuit jusqu'à ce qu'une solution optimale soit trouvée ou que le nombre d'itérations maximal a été atteint.

Une autre variante couramment utilisée est l'algorithme **Walksat** dérivant du GSAT et initialement introduit par [Selman et al., 1994]. A chaque itération, l'algorithme sélectionne aléatoirement une clause non satisfaite, et une variable de cette clause à flipper. Suivant une certaine probabilité (probabilité de bruit), il choisit entre deux façons de sélectionner la variable à flipper :

1. Une variable qui minimisera le nombre de clauses non satisfaites.

2. Une variable choisie aléatoirement pour la diversification afin d'échapper aux optimums locaux.

Récemment, de nouvelles recherches locales ont vu le jour, notamment [Xu et al., 2019].

Les métaheuristiques ont été largement utilisées pour la résolution du problème Max-SAT. La recherche tabou [Smyth et al., 2003], la recherche locale stochastique (SLS) [Hoos and Stützle, 2000, Marques-Silva and Sakallah, 1999, Mastrolilli and Gambardella, 2005, Selman et al., 1992], la recherche locale avec configuration checking [Luo et al., 2014] les algorithmes évolutionnaires (EA) [Gottlieb et al., 2002, Layeb and Saidouni, 2008, Marchiori and Rossi, 1999], la recherche dispersée [Boughaci et al., 2008], les algorithmes mémétique [Boughaci et al., 2004] et des méthodes hybrides entre EA et SLS ou méthodes exactes [Lardeux et al., 2006, Layeb et al., 2010]. Il existe aussi, des méthodes combinant la recherche locale et des méthodes exactes [Fernandes and Lourenço, 2007, Kroc et al., 2009]. D'autres recherches récentes se concentrent sur le pré-traitement des instances comme [Hireche et al., 2020], qui utilise les techniques de clustering sur l'ensemble des clauses.

A notre connaissance, il n'existe pas de travaux d'hyper-heuristiques dédiés au problème Max-SAT.

Etant confronté à des instances de problèmes de plus en plus grandes, les chercheurs sont toujours à la quête de méthodes de résolution plus performantes. La tendance actuelle est de créer des *méthodes hybrides* qui s'efforcent de tirer parti des avantages des différentes approches. Parmi les méthodes hybrides émergeantes, celles qui utilisent le paradigme multiniveau.

## 2.4 Le paradigme multiniveau

### 2.4.1 Définition

La méthode multiniveau, *multilevel* en anglais, dérive des méthodes multigrid utilisées en physique depuis les années 1970 pour résoudre des équations différentielles. Grâce aux travaux de Bruce Hendrickson et Robert Leland [Hendrickson and Leland, 1995] et de George Karypis et Vipin Kumar [Karypis and Kumar, 1995], elle est devenue populaire notamment pour le partitionnement de graphe (*Graph Partitioning Problem* (GPP)). Comme la méthode multiniveau est très performante pour résoudre des problèmes de GPP, elle a progressivement remplacé les méthodes spectrales (qui étaient les plus utilisées)

dans la seconde moitié des années 1990. Au début des années 2000, Chris Walshaw a étendu l'utilisation de cette méthode à d'autres problèmes d'optimisation combinatoire, comme le problème du voyageur de commerce (*Travelling Salesman Problem* (TSP)) [Walshaw, 2001b, Walshaw, 2002], la coloration de graphe (*Graph Coloring Problem* (GCP)) [Walshaw, 2001a], ou encore celui de la tournée de véhicules (*Vehicle Routing Problem* (VRP)) [Rodney et al., 2005].

### 2.4.2 Principe général

Le paradigme multiniveau est une contraction récursive qui crée une hiérarchie d'approximations (échantillons) du problème initial. Alors, une première solution est trouvée (au niveau le plus contracté). Puis une opération d'extension et d'affinement est itérée à chaque niveau. Des opérateurs de projection transfèrent alors la solution d'un niveau à un autre.

La méthode multiniveau se décompose en trois phases distinctes [Teng, 1999] (figure 2.1) :

- **Contraction (*coarsening step* ou *Up-Bottom Phase*)** : Elle est de nature itérative. Elle définit la structure hiérarchique ($P_0$, $P_1$, ..., $P_L$) de $L$ niveaux, pour un problème donnée $P_0 = P$ ;
- **Initialisation (*initialization step* ou *Basis Step*)** : Elle produit une solution initiale de $P_L$ ;
- **Extension et raffinement (*extension and refinement* step ou *Down-Top Phase*)** : l'étape qui construira une solution $S_i$ pour $P_i$ à partir d'une solution initiale $S_{i+1}$ de $P_{i+1}$ (niveau précédent).

L'algorithme 2 montre en pseudo code le principe général.

### 2.4.3 Applicabilité

Il est important de savoir si le paradigme multiniveau peut être appliqué facilement à n'importe quel problème. Pour cela il existe trois conditions : [Walshaw, 2008]

- Le problème doit avoir un algorithme d'affinement (une heuristique ou une métaheuristique) perturbatif. S'il n'en existe pas, c'est-à-dire les seuls algorithmes existant se basent sur la construction d'une solution, il n'est pas clair que le paradigme puisse être appliqué dans ce cas-là.
- l'algorithme d'affinement doit être capable de s'adapter à n'importe quelles restrictions imposées par l'algorithme de contraction (ex. dans le cas du partitionnement des graphes, les sommets du graphe contracté auront des poids même si initialement ils n'en possédaient pas).
- Il faut trouver une stratégie de contraction qui doit garder les mêmes valeurs de la fonction objectif et les mêmes propriétés du problème initiale à chaque étape. Cela garantit que cette procédure échantillonne le problème au lieu de le déformer.

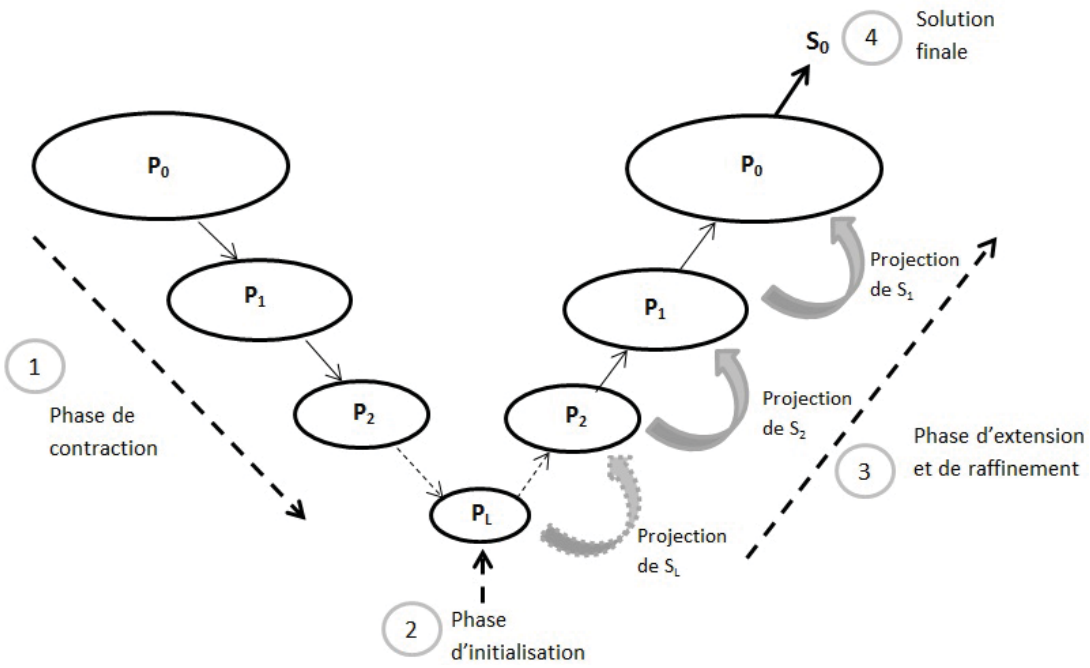



FIGURE 2.1 – Les phases de la méthode multiniveau

**Algorithm 2** Paradigme Multiniveau

**Entrée(s)** problème $P_0$.
**Sortie(s)** Solution $S_{final} = S_0$.

1: Niveau :=0 ;
2: // **Phase de Contraction**.
3: **Tantque** (Niveau < L) **faire**
4: $P_{Niveau+1}$ :=Contracter($P_{Niveau}$) ;
5: Niveau := Niveau+1 ;
6: **FinTantque**
7: // **Phase d'initialisation**.
8: $S_L$ :=solution_initiale($P_L$) ;
9: // **Phase de Raffinement**.
10: **Tantque** (Niveau > 0) **faire**
11: // **Extension du problème et projection de la solution du niveau précédent**
12: $S_{Niveau-1} := etendre(S_{Niveau})$ ;
13: $Niveau := Niveau - 1$ ;
14: // **Raffinement de la solution projetée**
15: $S_{Niveau}$ := Raffinement($S_{Niveau}$) ;
16: **FinTantque**
17: **Retourner** $S_{Niveau}$

#### 2.4.3.1 Exemple d'application au problème du voyageur de commerce

Elle a été établie pour la première fois en 2000 par [Walshaw and Cross, 2000], afin de montrer que le paradigme multiniveau peut être appliqué à d'autres

problèmes d'optimisation combinatoire autres que le partitionnement de graphe.

#### 2.4.3.1.1 Définition du problème

Le problème du voyageur de commerce est un problème classique d'optimisation : il s'agit de trouver la plus courte tournée permettant de visiter $n$ villes et de revenir au point de départ en ne visitant chaque ville qu'une seule fois. Ce problème se trouve généralement formulé dans le langage des graphes : on considère un graphe $(S, A)$ où $S$, les sommets du graphe, représentent les villes, et $A$, les arêtes, représentent les routes. Un cycle à $k$ sommets est alors un ensemble de $k$ sommets $(s_0, s_1, \ldots s_k)$ tels que $s_k = s_0$ et $(s_i, s_{i-1}) \in A$ pour $i = 1, 2, ...k$. La longueur d'un tel cycle est la somme des longueurs des arêtes qui le composent. Un cycle qui relie tous les sommets une et une seule fois est appelé cycle Hamiltonien. Le but du problème est de déterminer le cycle Hamiltonien de plus courte longueur.

#### 2.4.3.1.2 Phase de contraction [Walshaw, 2002]

L'algorithme de contraction fixe les arêtes de la tournée à chaque itération. Par exemple, soit un problème $P$ de taille $N$, si on fixe une arête reliant les deux villes $c_a$ et $c_b$ puis on crée un problème plus petit $P'$ de taille $N - 1$ (car il reste $N - 1$ arêtes à trouver) où la tournée trouvée de $P'$ doit obligatoirement contenir l'arête $(c_a, c_b)$.

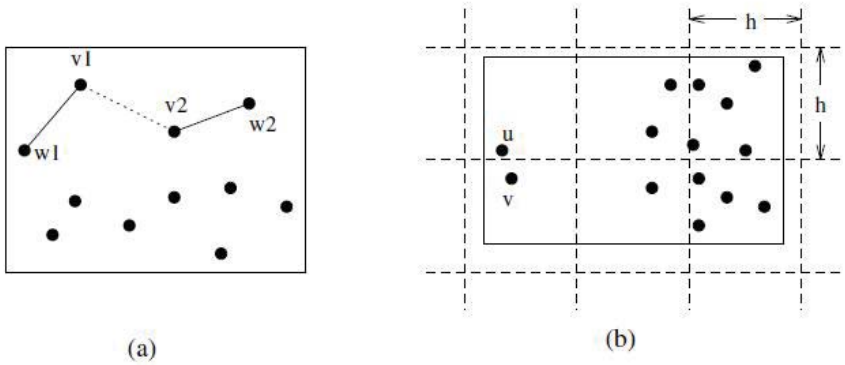



FIGURE 2.2 – Exemples de la stratégie de contraction pour TSP [Walshaw, 2002]

Le but de la stratégie de contraction est de fixer les arêtes qui ont plus de chance de figurer dans la solution finale. Par exemple dans la figure 2.2(b), il est difficile d'imaginer une tournée optimale qui n'inclut pas l'arête $(u, v)$, donc elle doit être fixée dans les premières étapes. En générale, les sommets doivent être fusionnés avec leurs voisins les plus proches. On fixe une distance maximale $h$ tel que seulement les sommets ayant une distance plus petite que $h$ peuvent être fusionnés. On calcule $h$ comme suit :

Si $A$ est la surface du rectangle d'aire minimum contenant tous les sommets, pour avoir en moyenne $n$ sommets dans chaque cellule (dont la surface est $h^2$), il faut que la surface de chaque cellule soit égale à $A_n/N$. Donc $h = \sqrt{A_n/N}$ , (figure 2.2(b)).

Les sommets $v_1$ et $v_2$ ayant été déjà fusionnés dans la figure 2.2(a), les sommets $w_1$ et $w_2$ ne doivent pas être fusionnés avec d'autres sommets, car cela peut contracter le problème trop rapidement.

Un exemple est donné sur la figure 2.3. La rangée supérieure montre le processus de contraction où les lignes en pointillées représentent les combinaisons des sommets (et par conséquent de nouvelles arêtes fixées) qui sont choisis dans l'itération courante, tandis que les lignes continues représentent les arêtes fixées lors des étapes de contraction précédentes. Notons qu'à partir de la troisième étape, des arêtes sont réduites à de simples sommets. L'étape de contraction se termine quand le problème est réduit à une arête fixée et deux sommets.

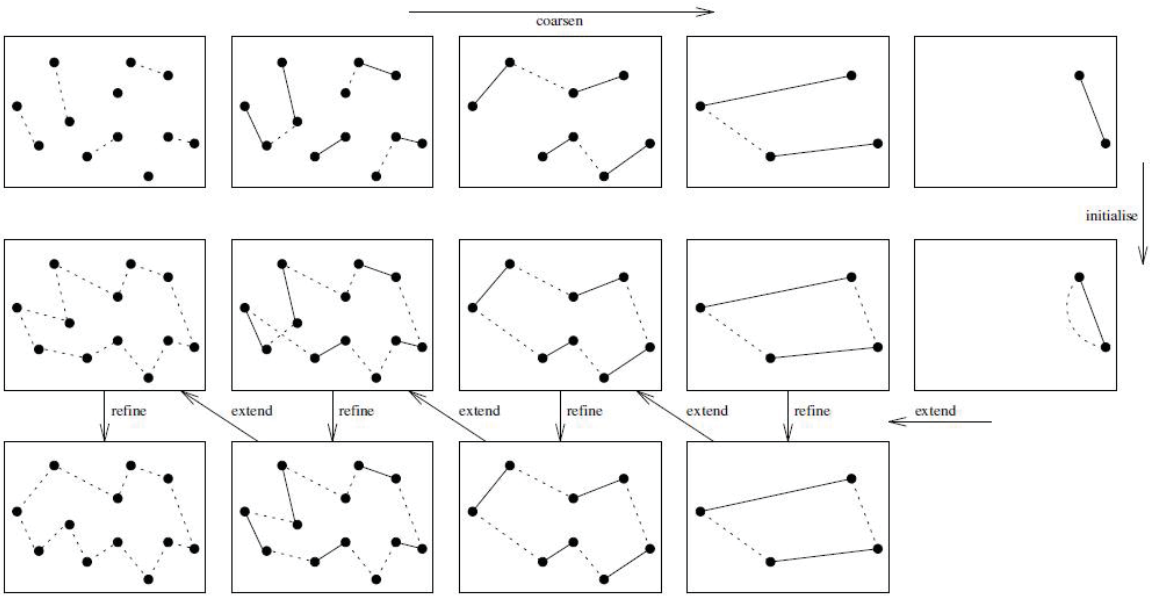



FIGURE 2.3 – Exemple d'exécution de la métaheuristique multiniveau sur une instance du problème TSP [Walshaw, 2002]

#### 2.4.3.1.3 Phase d'initialisation [Walshaw, 2002]

L'initialisation est insignifiante dans ce cas, et se réduit simplement à compléter le cycle en ajoutant une arête entre les deux sommets restants. Arrivant à cette étape nous pourrions juste déplier toutes les arêtes fixées et obtenir une première solution.

#### 2.4.3.1.4 Phase d'extension et de raffinement [Walshaw, 2002]

Cette phase est une boucle d'extension/raffinement (comme on peut le voir dans la deuxième et troisième rangée de la figure 2.3). Les lignes continues représentent les arêtes fixées tandis que les lignes en pointillés représentent les arêtes libres qui peuvent être changés par le raffinement. L'algorithme utilisé pour le raffinement peut être une heuristique ou une métaheuristique. Il essaye d'améliorer la solution (sans changer les arêtes fixées). La solution finale est montrée du côté gauche en bas de la figure 2.3.

### 2.4.4 Métaheuristiques multiniveau

L'algorithme de raffinement est généralement une simple stratégie de recherche locale. Cependant, il peut être plus sophistiqué tel que le recuit simulé ou les algorithmes génétiques [Walshaw, 2008]. D'où vient l'idée de l'hybridation des métaheuristiques.

Pour implémenter une métaheuristique multiniveau, nous avons besoin de quatre composants de base [Walshaw, 2008] :

- **Algorithme de contraction (*coarsening algorithm*)** qui contracte le problème récursivement ;
- **Algorithme d'initialisation (*initialisation algorithm*)** qui trouve une solution initiale au niveau le plus contracté ;
- **Algorithme d'extension (*extension algorithm*)** qui prend la solution du problème courant et la prolonge au problème parent ;
- **Algorithme de raffinement (*refienement algorithm*)** qui trouve la meilleure solution du problème courant (à partir d'une solution initiale du niveau précédent).

Les exemples existants du partitionnement des graphes regroupent quelques caractéristiques communes [Walshaw, 2008] :

- **L'algorithme de contraction**, qui est peut-être la composante clé de l'optimisation multiniveau, semble avoir trois principes :
  - **Principe 1** : N'importe quelle solution de n'importe quel espace contracté devrait inclure une solution légitime de l'espace original (même si la solution est loin d'être optimale). Ainsi, à n'importe quelle étape après l'initialisation, la solution courante pourrait simplement être prolongée à travers tous les niveaux du problème pour arriver à une solution du problème original. En outre les deux solutions (dans l'espace contracté et l'espace original) devraient avoir le même coût en ce qui concerne la fonction objectif. Cette condition assure que la contraction échantillonne l'espace de solution plutôt que de le déformer.
  - **Principe 2** : Le nombre de niveaux $L$ n'as pas besoin d'être déterminés à priori, mais la contraction devrait cesser quand tout autre itération rendrait l'initialisation dégénérée.
  - **Principe 3** : idéalement, n'importe quelle solution d'un niveau contracté devra avoir le même coût (par rapport à la fonction objective) que son extension au problème initial, ce qui veut dire que la contraction est exacte. Ce critère garantit que la contraction échantillonne vraiment l'espace de solution. Cependant, le paradigme semble fonctionner même si cette condition n'est pas remplie. De plus, une contraction exacte n'est pas toujours possible.

TABLE 2.1 – Références de travaux de métaheuristiques multiniveau [Walshaw, 2008]

| Métaheuristiques multiniveau | Références |
|---|---|
| Colonies de fourmis | [Langham and Grant, 1999, Korošec et al., 2004] |
| Recherche coopérative | [Toulouse et al., 1999] |
| Algorithmes génétiques | [Kaveh and Bondarabady, 2003, Soper et al., 2004] |
| Recherche taboue | [Battiti et al., 1999] ; Vanderstraeten et al., 1996] |
| Recuit simulée | [Romem et al., 1995] ; Vanderstraeten et al., 1996] |

TABLE 2.2 – Références récentes de recherches utilisant le paradigme multiniveau [Walshaw, 2008]

| Domaine d'application | Métaheuristiques | Références |
|---|---|---|
| Sélection de d'attributs biomédicale | Recherche taboue | [Oduntan, 2006] |
| Séquencement de l'ADN | Colonies de fourmis | [Blum et al., 2008] |
| Capacitated multicommodity network design | Recherche coopérative | [Crainic et al., 2006] |
| Problèmes d'optimisation numériques | Colonies de fourmis | [Korošec and Šilc, 2012] |
| Clustering (data mining) | Recherche taboue | [Rodney et al., 2008] |
| Problème SAT | Algorithme mémétique | [Bouhmala, 2012] |

Ceci ne nous indique toujours pas comment contracter un problème donné. Jusqu'ici la plupart des solutions pour le problème de partitionnement de graphe ont utilisé une réduction progressive et assez uniforme. En général, elle consiste à fusionner des groupes (souvent des paires) en une seule variable pour le niveau suivant.

- **L'initialisation :** La solution initiale est généralement «évidente» et l'algorithme d'affinage ne peut probablement pas l'améliorer au niveau le plus contracté (car il n'y a aucun degré de liberté).
  Exemples :
  GPP : affecter $k$ sommets restant à $k$ partitions,
  TSP : construire une tournée pour visiter deux villes, etc.

- **L'algorithme d'extension** doit être une inversion simple et évidente de l'étape de contraction qui préserve les mêmes paramètres.

- **L'algorithme d'affinage** peut être une heuristique ou une métaheuristique tel que : recuit simulé, recherche tabou, algorithme génétique, etc. Voici dans les tableaux 2.1 et 2.2 [Walshaw, 2008], quelques références de travaux dans le domaine des métaheuristiques multiniveau.

# Conclusion

Nous avons présenté dans ce chapitre le problème NP-difficile Max-SAT. Nous avons passé en revue les différentes méthodes de résolution exactes et approchées présente dans l'état de l'art. Nous avons présenté également le paradigme multiniveau. Nous avons détaillé son fonctionnement et ses compo-

santes de bases. Nous avons discuté les conditions de son applicabilité et la méthodologie à adopter afin de l'intégrer à des métaheuristiques. Et en fin, nous avons présenté quelques travaux de l'état de l'art utilisant ce paradigme.

Dans le chapitre suivant, nous présentons le deuxième problème auquel nous nous intéressons dans cette thèse, qui est le problème de sélection d'attributs dans la classification.

# Chapitre 3

# Problème de la sélection d'attributs en datamining

## 3.1 Introduction

Dans ce chapitre, nous introduisons le domaine de sélection d'attributs. Après avoir souligné l'intérêt de cette technique, nous détaillons les différentes phases de la sélection d'attributs. Puis, nous présentons les différentes techniques de l'état de l'art, notamment celle basées sur les métaheuristiques et hyper-heuristiques.

## 3.2 La sélection d'attributs

De nos jours, la croissance rapide d'Internet et des technologies a entraîné une croissance exponentielle des données récoltées, à la fois en dimensions et en taille. Confrontés à ces données, les algorithmes d'apprentissages, notamment la classification, deviennent de moins en moins efficaces. Dans des situations réelles, les attributs pertinents sont souvent inconnus à priori. De plus, l'existence des attributs non informatifs ou bruitées perturbent la classification, d'où l'intérêt de la phase de sélection d'attributs.

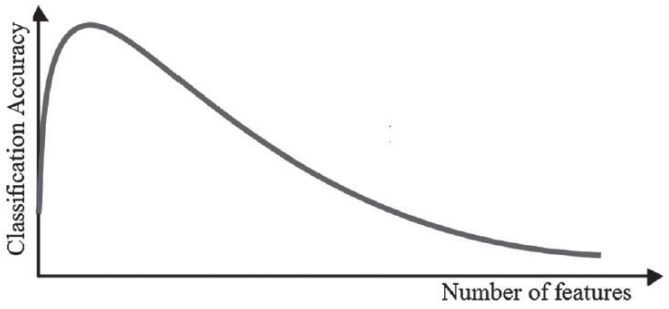



FIGURE 3.1 – L'impact du nombre d'attributs sur la précision de classification [Montazeri, 2016]

Un attribut est une variable qui décrit un aspect (dimension) d'une instance de donnée, par exemple, un rectangle possède deux attributs : longueur

et largeur. Un cube possède trois attributs : longueur, largeur et hauteur. Des instances de donnée plus complexe comme ceux décrivant des génomes peuvent avoir jusqu'à 60000 attributs. Ce nombre très grand rend le processus d'apprentissage beaucoup plus difficile. C'est pour cela, un pré-traitement qui sélectionne un sous-ensemble d'attributs les plus pertinents est nécessaire [Guyon and Elisseeff, 2003]. Comme indiqué dans la figure 3.1, pour une certaine taille de donnée, la précision de la classification augmente (non linéairement) par rapport au nombre d'attributs existants, jusqu'à arriver à un certain point critique, à partir duquel, l'augmentation du nombre d'attributs fait diminuer la précision de la classification. Donc, contrairement à ce qu'on aurait pensé, l'augmentation du nombre d'attributs ne garantit pas une meilleure classification, bien au contraire.

Donc la sélection d'attributs permet d'éliminer les attributs non pertinent, redondant ou bruité. Elle accélère le processus de classification, et améliore la précision de la prédiction. Elle facilite aussi la visualisation des données et la compréhension des résultats.

Formellement, nous pouvons définir le problème de la sélection d'attributs comme suit :
Soit $Y$ l'ensemble original des attributs de cardinalité $N$. Soit $f$ la fonction d'évaluation, tel que $f(X)$ représente la précision de la classification du sous-ensemble $X$, donc plus $f(X)$ est grand, plus $X$ est jugé meilleur. Le problème de la sélection d'attributs revient à trouver un sous-ensemble $X \subseteq Y$ tel que $|X| = d$ et :

$$f(X) = \max f(Z) \quad \text{où} \quad Z \subseteq Y, \quad |Z| = d, \quad d = 1..N$$

Le processus de sélection d'attributs peut être considéré comme un problème d'optimisation, où une solution est représentée par un vecteur de taille $N$ égale au nombre d'attributs de l'ensemble de donné (*dataset*). Chaque élément du vecteur peut prendre la valeur 1 si l'attribut correspondant est sélectionné, et 0 sinon [Kohavi and John, 1997]. Par conséquent, il y a un total de $2^N$ de sous ensembles.

Il est extrêmement difficile de trouver un sous-ensemble d'attributs optimal et de nombreux problèmes liés à la sélection d'attributs se sont révélés être NP-difficiles [Blum and Rivest, 1989].

## 3.3 Le processus de sélection d'attributs

Le processus de sélection d'attributs est composé de quatre étapes de base [Dash and Liu, 1997], comme illustré à la Figure 3.2 :

1. **Procédure de génération** : Qui génère les sous-ensembles d'attributs qui vont être évalués.
2. **Procédure d'évaluation** : Qui mesure la qualité du sous-ensemble généré.
3. **Critère d'arrêt** : Pour décider quand s'arrêter.

4. **Procédure de validation** : pour vérifier si le sous-ensemble d'attributs est valide.

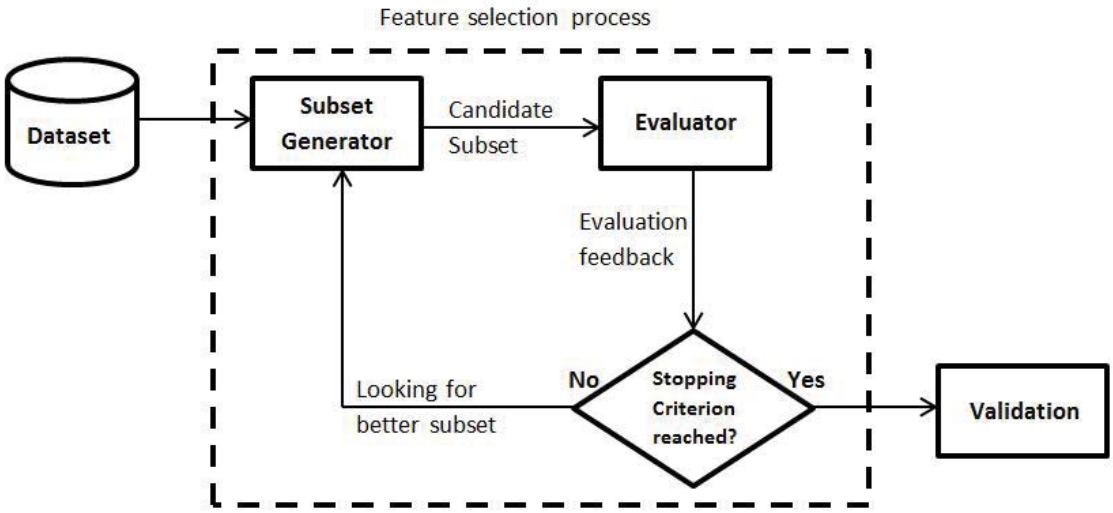



FIGURE 3.2 – Processus de sélection d'attributs

### 3.3.1 La procédure de génération

Il s'agit d'une procédure de recherche qui génère des sous-ensembles d'attributs à évaluer. Il existe trois types de procédures de génération :

1. Complète : Il s'agit d'une méthode complète qui commence par un ensemble vide, puis les attributs sont soit ajoutés, soit supprimés. Il s'agit d'une recherche exhaustive qui garantit le sous-ensemble optimal d'attributs. Cependant, la complexité de la recherche est exponentielle dans le pire des cas. Si N est le nombre d'attributs, alors la complexité est d'ordre $(2^N)$. Cela pourrait rendre cette méthode inutilisable en pratique lorsque le nombre d'attributs est supérieur à un certain seuil.

2. Heuristique : Il s'agit d'une méthode incomplète qui commence par un ensemble complet ou vide, puis ajoute ou supprime des attributs par rapport à une stratégie donnée afin d'obtenir un bon sous-ensemble d'attributs. Cela ne garantit pas la solution optimale, car il ne passe pas par tous les sous-ensembles possibles. Cependant, des solutions de bonne qualité pourraient être trouvées dans un délai raisonnable.

3. Aléatoire : Il commence avec un ensemble aléatoire d'attributs, puis les attributs sont soit ajoutées, soit supprimées de manière aléatoire.

### 3.3.2 La fonction d'évaluation

Elle mesure principalement la qualité du sous-ensemble d'attributs actuel généré par la procédure de génération, et la compare à la meilleure solution trouvée

jusqu'à présent. S'il est meilleur, le sous-ensemble actuel remplace le meilleur sous-ensemble précédent. Il existe trois stratégies d'évaluation des sous-ensembles d'attributs [Liu et al., 2009] :

1. Le modèle de filtre : Les approches par filtre sont considérées comme une étape de prétraitement de données. Cela signifie qu'elle évalue les attributs avant même l'application d'un algorithme d'apprentissage, ou même avant de décider quel classifieur utiliser. L'évaluation est effectuée en calculant des mesures qui permettent de sélectionner les attributs les plus pertinents. Cette approche présente deux avantages, la première est que la structure de l'algorithme est généralement simple et très rapide, ce qui le rend efficace même pour des tailles de données très grandes, la deuxième est qu'elle est complètement indépendante de l'algorithme d'apprentissage à utiliser. Par contre, l'optimalité du sous ensemble d'attributs sélectionnés ne garantit pas la meilleure performance du classifieur, en effet, comparées à cette méthode, les approches enveloppantes et intégrées donnent de meilleures résultats lors de l'apprentissage.

2. Le modèle enveloppant (wrapper) : Les approches enveloppantes évaluent les sous-ensembles d'attributs en utilisant un algorithme d'apprentissage. Ces méthodes permettent d'obtenir de bonnes performances. Cependant, elle nécessite pour chaque sous-ensemble d'attributs candidat d'effectuer la classification, ce qui peut devenir très couteux en temps de calcule surtout pour des données de dimension très grande.

3. Le modèle intégré : les approches intégrées essaient de combiner les avantages des deux approches précédentes (par filtre et enveloppantes). Dans cette approche, le processus de sélection est intégré au processus de classification comme dans SVM [Burges, 1998], adaboost [Schapire, 1999] ou CART [Breiman et al., 1984].

La figure 3.3 résume les caractéristiques des 3 approches.

### 3.3.3 Critère d'arrêt

Le processus de sélection d'attributs se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. [Dash and Liu, 1997] distinguent deux types de critères d'arrêt :

1. **Le critère d'arrêt basé sur une procédure de génération** : dans laquelle :
   - L'arrêt est exécuté lorsqu'un nombre prédéfini d'attributs est sélectionné.
   - L'arrêt est exécuté lorsqu'un nombre prédéfini d'itérations est atteint.

2. **Le critère d'arrêt basé sur une fonction d'évaluation** : où :
   - L'arrêt est exécuté lorsque l'ajout (ou la suppression) d'un attribut ne produit pas un meilleur sous-ensemble.
   - L'arrêt est exécuté lorsqu'un sous-ensemble optimal selon une fonction d'évaluation est obtenu.

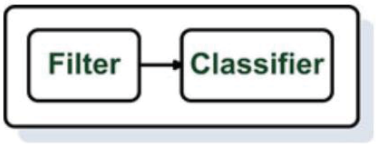
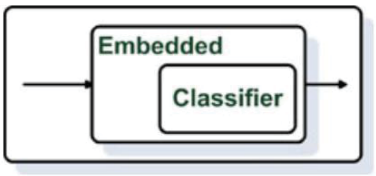
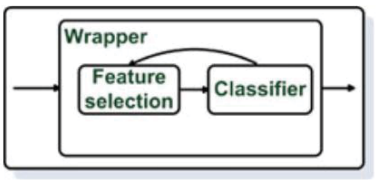
| Method | Advantages | Disadvantages | Examples |
|---|---|---|---|
| Filter   | Independence of the classifier<br>Lower computational cost than wrappers<br>Fast<br>Good generalization ability | No interaction with the classifier | Consistency-based CFS<br>INTERACT<br>ReliefF<br>$\mathcal{M}_d$<br>Information Gain<br>mRMR |
| Embedded   | Interaction with the classifier<br>Lower computational cost than wrappers<br>Captures feature dependencies | Classifier-dependent selection | FS-Percepton<br>SVM-RFE |
| Wrapper   | Interaction with the classifier<br>Captures feature dependencies | Computationally expensive<br>Risk of overfitting<br>Classifier-dependent selection | Wrapper-C4.5<br>Wrapper SVM |

FIGURE 3.3 – Technique de sélection d'attributs [Bolón-Canedo et al., 2013]

### 3.3.4 Une procédure de validation

La procédure de validation ne fait pas partie du processus de sélection d'attributs [Dash and Liu, 1997]. Il existe deux scénarios selon que nous avons ou non une connaissance préalable des données considérées. Le premier scénario consiste à connaître les attributs pertinents. Dans ce cas, ces attributs pourraient être comparés directement avec l'ensemble des attributs retenu par le processus de sélection d'attributs. Mais, dans les applications du monde réel, nous n'avons généralement pas de telles connaissances préalables. Il s'agit du deuxième scénario. Dans ce cas, la précision de l'ensemble d'attributs retenu est comparée en utilisant d'autres méthodes connues de la sélection d'attributs.

## 3.4 Méthodes de l'état de l'art

Nous distinguons deux classes de méthodes appliquées au problème de la sélection d'attributs :

1. **Les méthodes exactes** :

   (a) **Brench and Bound** :
   cette méthode a été utilisée par [Narendra and Fukunaga, 1977] pour la résolution du problème de la sélection d'attributs en 1977.

Elle construit un arbre de recherche où la racine représente l'ensemble des attributs et les autres nœuds représentent des sous-ensembles d'attributs. L'algorithme parcours l'arbre de la racine jusqu'au feuille en enlevant le plus mauvais attribut du sous-ensemble courant qui ne satisfait pas le critère de sélection. Une fois que la valeur attribuée à un nœud est inférieur à un seuil, les sous arbres de ce nœud sont supprimés.

(b) **FOCUS** : En 1991 [Almuallim and Dietterich, 1991] ont proposé une méthode de filtrage pour la sélection d'attributs appelée *FOCUS*. Cette méthode se base sur une recherche exhaustive de sous-ensemble d'attributs. L'algorithme commence par évaluer chaque sous-attribut de taille 1, puis évalue tous les couples d'attributs, puis les triplets d'attributs, ainsi de suite jusqu'à arriver à un critère d'arrêt. L'algorithme ne gère pas efficacement les données bruitées, et deviens inutilisable si la taille des données en entrées est très grande.

2. Les méthodes approchées :

(a) **Les méthodes de recherche heuristique** :

i. **Sélection séquentielle arrière** :
La sélection séquentielle arrière ou *Sequential Backward selection* (SBS) a été proposé par [Marill and Green, 1963]. C'est une approche heuristique qui, contrairement à SFS, commence par un ensemble complet d'attributs, et qui à chaque itération, enlève les attributs un à un en sélectionnant le moins pertinent selon des mesures statistiques. Le processus est réitéré jusqu'au critère d'arrêt. Cependant, cette méthode possède un inconvénient majeur, car enlever à chaque étape l'attribut qui est localement le moins pertinent ne pourra pas être corrigé par la suite. Ce qui conduira la recherche vers un optimum locale où elle sera coincée.

ii. **Sélection séquentielle croissante** : La sélection séquentielle croissante ou *Sequential Forward selection* (SFS) a été proposé par [Whitney, 1971]. C'est une approche heuristique qui commence par un ensemble vide d'attributs, et qui à chaque itération, rajoute les attributs un à un en sélectionnant le plus pertinent selon des mesures statistiques. Le processus est réitéré jusqu'au critère d'arrêt. Cette méthode a le même inconvénient que la précédente, où la méthode tombe dans un optimum locale.

Les deux méthodes ont été améliorées en 1994 par [Pudil et al., 1994a, Pudil et al., 1994b] qui a introduit la notion de retour arrière (*Backtracking*) afin de sortir des optimums locaux.

(b) **Les métaheuristiques** :
Plusieurs métaheuristiques ont été implémentées pour le problème de la sélection d'attributs. Les méthodes évolutionnaires ont été

très utilisées pour la résolution de ce problème notamment dans [Tsai et al., 2013, Moradi and Rostami, 2015, Wang et al., 2017, Jain and Zongker, 1997, Ishibuchi and Nakashima, 2000, Kuncheva and Jain, 1999]. La métaheuristique de colonie de fourmi (AntRSAR) et la métaheuristique génétique (GenRSAR) sont deux méthodes évolutionnaires qui ont été proposé par [Jensen and Shen, 2003] pour le problème de la sélection d'attributs. Une méthode hybride combinant l'algorithme génétique et le recuit simulé a été proposée dans [Mafarja and Abdullah, 2013], un algorithme mémétique a été utilisé dans [Nekkaa and Boughaci, 2015]. Une recherche dispersée (*Scatter Search*) SSAR a été proposée par [Wang et al., 2007] dans une méthode de sélection d'attributs basée algorithme évolutionnaire. Des métaheuristiques inspirées de la nature ont été aussi utilisées, par exemple nous pouvons citer *Ant Lion Optimizer* (ALO) [Emary et al., 2016, Zawbaa et al., 2015]. Un algorithme d'optimisation par essaim de particule (PSO) a été proposé par [Unler and Murat, 2010, Lin et al., 2008]. Des algorithmes d'essaim d'abeilles ont aussi été utilisés dans [Schiezaro and Pedrini, 2013, Shanthi and Bhaskaran, 2014, Palanisamy and Kanmani, 2012]. D'autres algorithmes de colonie de fourmi ont été proposés dans [Kashef and Nezamabadi-pour, 2015, Kabir et al., 2012, Aghdam et al., 2009, Al-Ani, 2005]. Des algorithmes mimétique ont été appliqués avec succès dans [Yang et al., 2008, Lee and Kim, 2015, Zhu et al., 2007]. Les algorithmes de recherche locale stochastique ont été aussi utilisés pour ce problème [Nekkaa and Boughaci, 2016, Chebouba et al., 2018].

(c) **Les hyper-heuristiques** :

Les hyper-heuristiques ont été très peu utilisées pour ce problème. Le plus récent est celui de [Montazeri, 2016], qui proposent deux hyper-heuristiques appelées *Hyper-Heuristic Feature Selection* (HHFS) et *Adaptive Hyper-Heuristic Feature Selection* (AHHFS) pour la résolution du problème de la sélection d'attributs. Ces deux hyper-heuristiques se basent sur une stratégie d'algorithme génétique, afin de sélectionner des séquences d'heuristiques de bas niveau appelées « **chromosome** ». Les deux hyper-heuristiques procèdent de la même manière, mais la différence réside dans la taille des chromosomes. Dans HHFS, la taille des chromosomes est fixe, par contre dans AHHFS la taille des chromosomes est variable, et elle peut changer d'une génération à une autre. Comme montré dans l'algorithme 3, l'hyper-heuristique commence par générer aléatoirement une population de chromosomes d'heuristiques de bas niveau. Elle génère une solution initiale en sélectionnant aléatoirement un sous-ensemble d'attributs. Ensuite, à chaque itération, chaque chromosome est appliqué à la solution courante, et la solution retournée par la séquence des heuristiques de bas niveau est évaluée par la fonction objectif. La meilleure solution est alors sélectionnée, puis des

opérations de sélection, mutation et croisement sont appliquées sur population afin de produire une nouvelle génération pour l'itération suivante. Le processus est réitéré jusqu'à atteindre un critère d'arrêt. La figure 3.4 illustre le déroulement de l'algorithme HHFS.

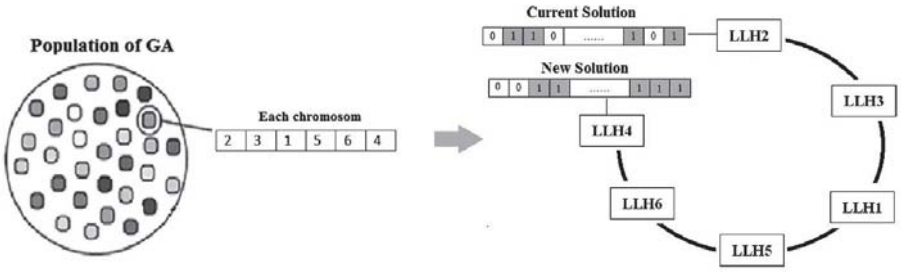



FIGURE 3.4 – Processus de HHFS [Montazeri, 2016]

[Montazeri, 2016] a utilisé le classifier 1-NN afin d'évaluer les ensembles d'attributs sélectionnés. 16 heuristiques de bas niveau ont été utilisées, 12 d'entre elles sont des recherches locales d'intensification similaires aux algorithmes Hill-climbing, qui sont des méthodes itératives et gloutonnes. Les 4 heuristiques de bas niveau restantes sont des méthodes de diversification qui se basent sur des opérations de croisement et de mutation. Les résultats ont été comparés avec 4 méthodes de l'état de l'art, la première méthode basée filtre appelée DMIFS [Liu et al., 2009], la seconde est une méthode enveloppante qui utilise une métaheuristique [Tahir and Smith, 2010], la troisième appelée *Bagging Constraints Score* (BCS) [Sun and Zhang, 2010] qui est une méthode basée filtre et enveloppante en même temps, la dernière méthode est une méthode basée filtre appelée CoFS [Sun et al., 2012]. Les résultats ont montré que HHFS est meilleure que l'AHHFS et les 4 méthodes de l'état de l'art.

**Algorithm 3** Hyper-Heuristic Feature Selection (HHFS)

1: Générer aléatoirement une population de chromosomes d'heuristiques de bas niveau.
2: Générer aléatoirement une solution initiale (ensemble d'attributs).
3: **Tantque** (critère d'arrêt non atteint) **faire**
4: **Pour Chaque** chromosome de la population **faire**
5: Appliquer le chromosome sur la solution courante.
6: Sauvegarder la nouvelle solution $fs_i$
7: **FinPour**
8: Evaluer toute les solutions des chromosomes par la fonction objectif.
9: Sélectionner la meilleure solution.
10: Appliquer les opérateurs de sélection, mutation et croisement.
11: **FinTantque**

## 3.5 Conclusion

Nous avons abordé dans ce chapitre le problème de sélection d'attributs. Nous avons exposé le processus suivit pour la résolution de ce problème, nous avons décrit quelques méthodes de l'état de l'art pour la sélection d'attributs en s'intéressant surtout aux les méthodes basées métaheuristiques et hyper-heuristiques.

Malgré quelques progrès, les techniques de sélection d'attributs ne sont pas encore complètement satisfaisantes. Elles sont soit applicables sur de grandes instances mais ne donnent pas de résultats optimaux, ou bien donnent des résultats optimaux ou presque optimaux, mais ne peuvent pas faire face à la complexité des grandes instances des données du monde réel. Des méthodes de sélection d'attributs plus efficaces (donnant une solution optimale et pouvant être appliquées sur de grandes instances du monde réel) restent à être développées.

# Chapitre 4

# Approches Hyper-heuristiques et paradigme multiniveau pour le problème Max-SAT

## 4.1 Introduction

Dans ce chapitre, nous proposons tout d'abord une approche hyper-heuristique appelée stochastique choice function pour le problème Max-SAT. L'approche proposée est basée sur le mécanisme d'apprentissage choice function et l'aléatoire. Dans le but de mesurer sa performance, une comparison avec l'hyper–heuristique choice function et l'hyper-heuristique de sélection aléatoire est établie dans la première partie du chapitre.

Dans la seconde partie, nous introduisons le paradigme multiniveau dans une nouvelle approche hyper-heuristique, et ce dans le but de résoudre les instances de grande taille de problème Max-SAT. Nous validons notre approche sur des instances de petite, moyenne et grande taille. Une comparison avec l'état de l'art est ajoutée pour montrer l'efficacité de notre approche.

## 4.2 Approches hyper-heuristiques pour Max-SAT

Dans ce qui suit, nous allons décrire, tout d'abord les composantes essentielles des trois hyper-heuristiques que nous avons proposées pour le problème Max-SAT. Les trois approches diffèrent par la méthode de sélection des heuristiques de bas niveau.

Ensuite nous présentons les trois approches proposées à savoir : L'hyper-heuristique aléatoire, l'‘hyper-heuristique de choice function et l'hybridation entre l'aléatoire et la méthode choice function.

Nous tenons à préciser que les trois hyper-heuristiques proposées acceptent toutes les solutions au cours du processus de recherche, cela permet de garantir une meilleure diversification.

### 4.2.1 Composantes des hyper-heuristiques

Dans ce qui suit, nous allons décrire les composantes des trois hyper-heuristiques testées. Elles diffèrent par la méthode de sélection des heuristiques de bas niveau. La première est l'hyper-heuristique aléatoire, la deuxième est l'hyper-heuristique choice function et la troisième est une hybridation entre l'aléatoire et la méthode informée choice function.

Dans notre cas les hyper-heuristiques proposées acceptent toutes les solutions au cours du processus de recherche, cela permet de garantir une meilleure diversification.

#### 4.2.1.1 Codification de la solution

Une solution est représentée par un vecteur binaire $X$ de taille $n, n$ étant le nombre de variables de l'instance. Chaque case du vecteur $X_i$ peut recevoir la valeur 0 (pour *faux*) ou bien 1 (pour *vrai*). Ce vecteur représente donc une affectation de valeurs de vérité aux $n$ variables de l'instance Max-SAT en cours de résolution. La qualité d'une solution (*fitness*) est mesurée par une fonction objectif qui consiste à minimiser le nombre de clauses insatisfaites.

#### 4.2.1.2 La solution initiale

La solution initiale est générée en affectant aléatoirement des valeurs de vérité aux variables du vecteur solution comme montré dans la Figure 4.1.

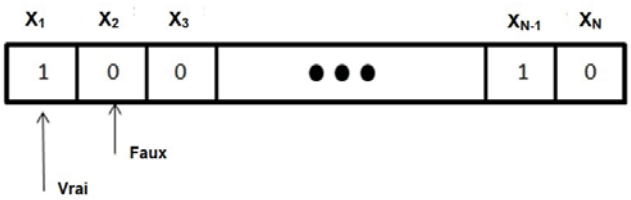



FIGURE 4.1 – Représentation d'une solution

#### 4.2.1.3 La fonction objectif

Dans le problème Max-SAT, une solution $X$ est évaluée en comptant le nombre de clauses insatisfaites dans l'instance. La solution optimale est celle qui obtient le nombre minimum. En d'autres termes, le but est de trouver une affectation de variables qui minimise le nombre de clauses insatisfaites dans l'instance. D'une manière formelle, la fonction objectif $f$ à minimiser est définie comme suit :

$$f(X) = \sum_{i=0}^{n} C_i \quad \text{où la clause} \quad C_i = \begin{cases} 1 & \text{si elle est satisfaite} \\ 0 & \text{sinon.} \end{cases}$$

#### 4.2.1.4 Les sept heuristiques de bas niveau pour Max-SAT

Sept heuristiques de bas niveau perturbatrices de stratégies différentes sont utilisées durant le processus de recherche. Certaines d'entre elles utilisent les informations d'une matrice de taille $n * 2, n$ étant le nombre de variables de l'instance. Cette matrice contient le nombre d'occurrence de chaque variable $x_i$ et de sa forme négative ($\bar{x_i}$) dans l'instance. Dans ce qui suit, nous donnons une description de chacune d'elles :

- L'heuristique $h_1$ : Cette heuristique applique une mutation sur la meilleure solution trouvée, en flippant une seule variable choisie selon les étapes suivantes :
  - Une variable est choisie au hasard, si elle est vraie (resp. fausse) dans la solution et que son nombre d'occurrences de sa forme négative est supérieur (resp. inférieur) au nombre d'occurrences de sa forme positive, alors elle sera flippée.
  - Sinon, choisir aléatoirement une variable, à partir d'un vecteur contenant celles dont le *num_improve* est positive. *num_improve* étant le nombre de clauses satisfaites moins le nombre de clauses insatisfaites si on flip la variable.
  - S'il n'y a aucune variable dont le *num_improve* est positive, alors une variable est choisie au hasard dans $n$.
- L'heuristique $h_2$ : Cette heuristique renvoie une nouvelle solution, en combinant point par point la solution courante, avec une solution construite, en utilisant la matrice décrite précédemment, comme suit :

$$V[x_i] = \begin{cases} 1 & \text{si occurence}(x_i) > \text{occurence}(\bar{x_i}) \\ 0 & \text{sinon.} \end{cases}$$

- L'heuristique $h_3$ :C'est une recherche locale stochastique qui explore le voisinage d'une solution candidate selon trois stratégies contrôlées par deux probabilités **walk1** et **walk2** comme suit :
  - Si probabilité est inférieure à **walk1**, alors choisir une variable aléatoirement et la flipper.
  - Sinon si probabilité est inférieure à **walk2**, alors choisir aléatoirement une clause insatisfaite et flipper la variable qui va satisfaire le plus de clauses.
  - Sinon, flipper la variable (parmi tous l'ensemble de variables) qui va satisfaire le plus de clauses.
- L'heuristique $h_4$ : Elle effectue une mutation sur la solution courante. La mutation se fait en choisissant une variable aléatoirement et en la flippant selon une certaine probabilité.

- L'heuristique $h_5$ : Cette heuristique combine la meilleure solution trouvée avec une solution générée aléatoirement, en faisant un croisement point par point selon une certaine probabilité.
- L'heuristique $h_6$ : Comme l'heuristique $h_4$, mais elle est répétée $n$ fois.
- L'heuristique $h_7$ : Cette heuristique prend la meilleure solution trouvée puis choisis la variable qui augmentera le plus le nombre de clauses satisfaites.

Une recherche locale simple est toujours appliquée à la fin de l'exécution de chaque heuristique de bas niveau, afin d'optimiser la solution courante. Cette méthode explore le voisinage de la solution retournée par l'heuristique de bas niveau avec une probabilité de marche (*walking probability*). Par contre, l'heuristique $h_3$ n'en a pas besoin puisqu'elle-même est une recherche locale.

### 4.2.2 l'hyper-heuristique aléatoire (R-HH)

#### 4.2.2.1 Principe

La stratégie de sélection aléatoire, comme son nom l'indique, choisie la prochaine heuristique de bas niveau à exécuter de manière aléatoire. Cette hyper-heuristique 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.

#### 4.2.2.2 Algorithme

L'algorithme 4 représente en pseudo-code l'hyper-heuristique aléatoire R-HH.

### 4.2.3 l'hyper-heuristique choice function (CF-HH)

#### 4.2.3.1 Principe

La stratégie choice function est une stratégie d'apprentissage par renforcement basée sur un score calculé comme suit :

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

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

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

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

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

Où :

- $h_i$ : une heuristique de bas niveau.

**Algorithm 4** Random Hyper-heuristique (R-HH)

**Entrée(s)** Une instance Max-SAT, Un ensemble d'heuristiques de bas niveau, le parametre maxiter.

**Sortie(s)** Une solution S.

1: Generer une solution initiale $S$.
2: $F :=$ Evaluation de la solution $S$.
3: $S^* = S; F^* = F$; // $S^*$ est la meilleure solution trouvé, $F^*$ represente sa qualité.
4: I= 0 ;
5: **Tantque** (I<maxiter) **faire**
6: **//Heuristique de selection**
7: $h_i :=$ choisir une heuristique de bas niveau aléatoirement.
8: Appliquer l'heuristique de bas niveau $h_i$ sur S, pour obtenir un nouveau S avec une qualité F.
9: **//Methode d'acceptation : Toute les solutions (All moves)**
10: **Si** $F < F^*$ **Alors**
11: $S^* := S; F^* := F$ ;
12: **Finsi**
13: **FinTantque**
14: **Retourner** $S^*$

- $h_{ID}$ : la dernière heuristique de bas niveau appelée.
- $I_n(h_i)$ (resp. $I_n(h_{ID}, h_i)$) : le changement de la fonction d'évaluation après la nième exécution de $h_i$ (resp. la $n^{\text{ème}}$ exécution de $h_i$ après $h_{ID}$).
- $T_n(h_i)$ (resp. $T_n(h_{ID}, h_i)$ ) : le temps d'exécution de $h_i$ après le nième appel (resp. le temps d'exécution du $n^{\text{ème}}$ appel de $h_i$ qui suit $h_{ID}$).
- $\alpha$,$\beta$ et $\delta$ : paramètres fixées empiriquement.

L'équation $g_1(h_i)$ permet d'évaluer la performance individuelle de l'heuristique de bas niveau $i$, en prenant en compte ses $n$ dernières exécutions. 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 $g_2(h_{ID}, h_i)$ mesure la synergie entre les heuristiques de bas niveau ($h_{ID}$ représente la dernière heuristique de bas niveau exécuté). L'équation $g_3(h_i)$ calcule le temps écoulé depuis la dernière exécution de l'heuristique de bas niveau $i$. Enfin, $score(h_i)$ collecte les résultats des 3 fonctions présentées précédemment et calcule le score de chaque heuristique de bas niveau.

#### 4.2.3.2 Algorithme

L'algorithme 5 représente en pseudo-code l'hyper-heuristique aléatoire CF-HH.

**Algorithm 5** Choice-Function Hyper-heuristique (CF-HH)

**Entrée(s)** Une instance Max-SAT, Un ensemble d'heuristiques de bas niveau, maxiter, $\alpha$, $\beta$, $\delta$.
**Sortie(s)** Une solution S.

```
1: Generer une solution initiale S.
2: F := Evaluation de la solution S.
3: S* = S; F* = F; // S* est la meilleure solution trouvé, F* represente sa qualité.
4: I= 0 ;
5: Tantque (I<maxiter) faire
6:    //Heuristique de selection
7:    h_i := choisir une heuristique de bas niveau ayant le plus grand score de choice function.
8:    Appliquer l'heuristique de bas niveau h_i sur S, pour obtenir un nouveau S avec une qualité F.
9:    //Methode d'acceptation : Toute les solutions (All moves)
10:   Mettre à jour les scores de chaque heuristique de bas niveau.
11:   Si F < F* Alors
12:      S* := S; F* := F ;
13:   Finsi
14: FinTantque
15: Retourner S*
```

### 4.2.4 Une approche hyper-heuristique Stochastique Choice-Function SCF-HH pour Max-SAT

#### 4.2.4.1 Principe

Cette hyper-heuristique utilise un module de sélection d'heuristiques de bas niveau hybride. Elle combine la méthode de sélection basée sur l'apprentissage par renforcement choice function, avec la méthode de sélection aléatoire.

L'hyper-heuristique hybride utilise un principe similaire à celui de la recherche locale stochastique. La méthode de sélection se base sur la stratégie choice function et l'aléatoire selon une probabilité fixée $wp > 0$.

#### 4.2.4.2 Algorithme SCF-HH pour Max-SAT

L'algorithme 6 représente en pseudo-code l'hyper-heuristique aléatoire SCF-HH.

### 4.2.5 Résultats expérimentaux

Pour l'étude expérimentale, nous avons implémenté, dans un premier temps, trois variantes d'hyper-heuristiques pour le problème Max-SAT, à savoir : l'**hyper-heuristique aléatoire** (**R-HH**), l'**hyper-heuristique choice function** (**CF-HH**) et l'**hyper-heuristique stochastique choice function** (**SCF-HH**). Toutes les méthodes ont été implémentées en langage C. Les programmes

**Algorithm 6** Stochastique Choice-Function Hyper-heuristique (SCF-HH)

**Entrée(s)** Une instance Max-SAT, Un ensemble d'heuristiques de bas niveau, maxiter, la probabilité wp, $\alpha$, $\beta$, $\delta$.

**Sortie(s)** Une solution S.

1: Generer une solution initiale $S$.
2: $F :=$ Evaluation de la solution $S$.
3: $S^* = S; F^* = F$; // $S^*$ est la meilleure solution trouvé, $F^*$ represente sa qualité.
4: I= 0 ;
5: **Tantque** (I<maxiter) **faire**
6: **//Heuristique de selection**
7: r= nombre aléatoire entre 0 et 1 ;
8: **Si** (r<wp) **Alors**
9: $h_i :=$ choisir une heuristique de bas niveau aléatoirement.
10: **Sinon**
11: $h_i :=$ choisir une heuristique de bas niveau ayant le plus grand score de *choice function*.
12: **Finsi**
13: Appliquer l'heuristique de bas niveau $h_i$ sur S, pour obtenir un nouveau S avec une qualité F.
14: **//Methode d'acceptation : Toute les solutions (All moves)**
15: Mettre à jour les scores de chaque heuristique de bas niveau.
16: **Si** $F < F^*$ **Alors**
17: $S^* := S; F^* := F$ ;
18: **Finsi**
19: **FinTantque**
20: **Retourner** $S^*$

ont été testés sur une machine avec un processeur intel i7 et 8GB de RAM.

#### 4.2.5.1 Description des benchmarks

Nous avons testé sur les benchmarks suivants :

- Benchmarks **Hoos** : Un ensemble de 25 instances SAT avec 250 variables et 1065 clauses pour chacune d'elles. Comme la taille de ces instances est relativement petite, nous avons fixé la limite du temps d'exécution à 300s.

- Benchmarks **bmc** : ils contiennent 10 instances SAT difficiles à résoudre ayant un nombre de variables qui varie entre 8710 et 59056, et un nombre de clauses qui varie entre 39774 et 323700 clauses. Dans ce cas, la limite du temps d'exécution a été fixée à 1800s.

- Benchmarks industriels de la compétition **Max-SAT 2014** : nous avons testé sur 54 instances UNSAT de très grandes tailles donc le nombre de variables atteint les 4 millions et le nombre de clauses les 15 millions. Dans ce cas aussi la limite du temps d'exécution a été fixée à 1800s.

#### 4.2.5.2 Phase de paramétrage

Nous avons fixé d'une manière empirique les paramètres suivants. Nous avons fait des tests rigoureux sur les benchmarks décrits précédemment, et les valeurs retenues sont celles qui donnent de bonnes performances.

- $\alpha$=0.9, $\beta$=0.1 et $\delta$=1.5.
- La valeur de **wp**, le paramètre qui gère le choix de la stratégie de sélection a été fixé à 0.4.

D'autres paramètres concernent les heuristiques de bas niveau :

- Probabilités pour $h_3$ (SLS) : **walk1** =0.3 et **walk2**=0.6.
- Degrés de mutation dans $h_4$ : **TM1**=0.152,
- Degrés de mutation dans $h_6$ : **TM2**= 0.071.

#### 4.2.5.3 Résultats

Les résultats obtenus sont détaillés à travers des tables et des figures, où la mesure principale est le pourcentage des clauses satisfaites par rapport au nombre de clauses de l'instance traitée. La meilleure solution de l'instance Max-SAT, qui est en gras, est celle avec le plus haut pourcentage.

##### 4.2.5.3.1 Les résultats du benchmark Hoos

Les résultats sur les benchmarks Hoos sont détaillés dans la table 4.1. Dans la table 4.2, les mesures statistiques sont exposées. Pour chaque méthode on donne le minimum (*min*), le maximum (*max*), la moyenne (*mean*), la médiane (*median*), le premier quartile (*1er Qu*) et le troisième quartile (*3eme Qu*). La figure 4.2 illustre le diagramme box plot afin de mieux visualiser la distribution des valeurs des taux de satisfactions pour chaque méthode.

Nous pouvons voir que la SCF-HH donne de bien meilleurs résultats que la simple CF-HH. Cependant, les résultats de SCF-HH se rapprochent de celle de R-HH.

##### 4.2.5.3.2 Les résultats du benchmark bmc

Les résultats sur les benchmarks bmc (*bounded model cheking*) sont détaillés dans la table 4.3 où le meilleur résultat est en gras. Les résultats de l'étude statistique sont présentés dans la table 4.4. Et pour une meilleure représentation, un box plot est montré à la figure 4.3. Les benchmarks bmc sont des instances de taille moyenne avec deux instances qui sont connus pour être difficiles à résoudre : galileo-8 et galileo-9.

Nous constatons des résultats similaires au résultat de Hoos, où SCF-HH et R-HH se rapprochent et dépassent CF-HH.

TABLE 4.1 – Resultats pour les benchmarks Hoos (SAT)

| **Benchmark** | **SCF-HH** | **CF-HH** | **R-HH** |
|---|---|---|---|
| Uf250-01 | **99.91**% | 99.34% | **99.91%** |
| Uf250-02 | **100.00%** | 99.44% | 99.81% |
| Uf250-03 | 99.62% | 99.44% | **99.72%** |
| Uf250-04 | **99.91%** | 99.72% | **99.91%** |
| Uf250-05 | **99.72%** | 99.53% | 99.62% |
| Uf250-06 | 99.72% | 99.44% | **99.81%** |
| Uf250-07 | 99.62% | 98.78% | **99.91%** |
| Uf250-08 | 99.72% | 99.62% | **99.81%** |
| Uf250-09 | **99.81%** | 98.78% | 99.53% |
| Uf250-010 | **99.81%** | 99.15% | **99.81%** |
| Uf250-011 | **99.81%** | 99.34% | 99.62% |
| Uf250-012 | **100.00%** | 99.72% | **100.00%** |
| Uf250-088 | **99.91%** | 99.15% | 99.81% |
| Uf250-089 | 99.62% | 99.15% | **100.00%** |
| Uf250-090 | 99.62% | 99.44% | **99.81%** |
| Uf250-091 | 99.91% | 99.44% | **100.00%** |
| Uf250-092 | 99.53% | 99.34% | **99.72%** |
| Uf250-093 | **99.81%** | 99.44% | **99.81%** |
| Uf250-094 | 99.72% | 99.53% | **99.81%** |
| Uf250-095 | 99.91% | 99.34% | **100.00%** |
| Uf250-096 | **99.81%** | 99.25% | 99.72% |
| Uf250-097 | **99.91%** | 99.44% | **99.91%** |
| Uf250-098 | **99.91%** | 99.34% | 99.81% |
| Uf250-099 | **99.91%** | 99.15% | 99.72% |
| Uf250-0100 | **99.72%** | 99.34% | **99.72%** |

TABLE 4.2 – Résultats statistiques du benchmark Hoos

| Methods | **Min** | **1st. Qu** | **Median** | **Mean** | **3rd Qu** | **Max** |
|---|---|---|---|---|---|---|
| **SCF-HH** | 99.53% | 99.72% | 99.81% | 99.80% | 99.91% | 100.00% |
| **CF-HH** | 98.78% | 99.25% | 99.34% | 99.35% | 99.44% | 99.72% |
| **R-HH** | **99.53%** | 99.72% | 99.81% | **99.81%** | 99.91% | 100.00% |

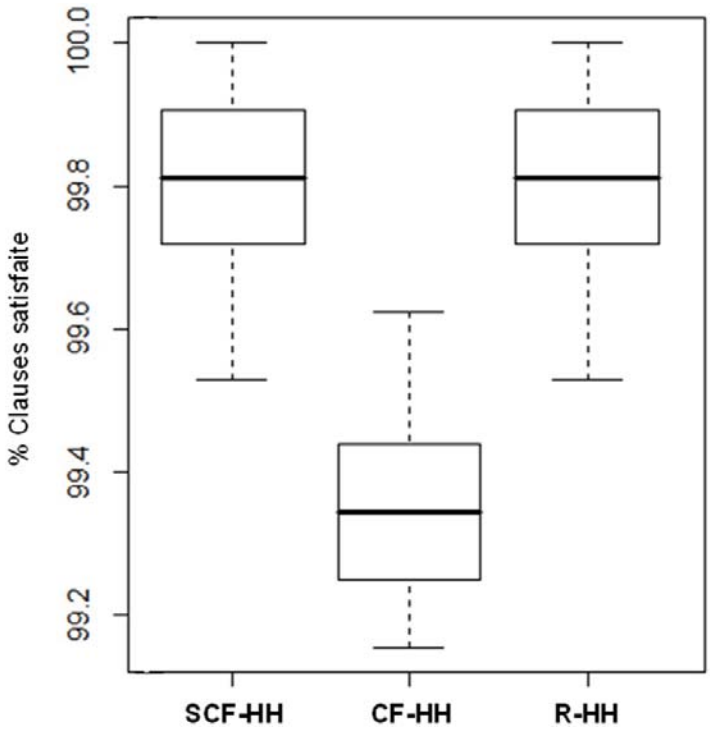



FIGURE 4.2 – Box-plot des résultats de SCF-HH, CF-HH et R-HH pour les benchmarks Hoos

TABLE 4.3 – Résultats du benchmark bmc (SAT)

| Benchmark | SCF-HH | CF-HH | R-HH |
|---|---|---|---|
| bmc-ibm-7 | **99.29%** | 96.74% | **99.26%** |
| bmc-ibm-5 | 99.43% | 96.37% | **99.75%** |
| bmc-ibm-1 | 99.38% | 99.12% | **99.56%** |
| bmc-ibm-13 | 98.66% | 97.18% | **99.19%** |
| bmc-ibm-3 | 98.39% | 96.86% | **99.06%** |
| bmc-ibm-11 | 96.60% | 95.88% | **98.26%** |
| bmc-ibm-12 | 96.28% | 95.48% | **97.62%** |
| bmc-galileo-8 | 92.90% | 92.01% | **95.93%** |
| bmc-ibm-10 | 96.00% | 95.84% | **97.04%** |
| bmc-galileo-9 | 92.62% | 91.98% | **94.20%** |

#### 4.2.5.3.3 Les résultats du benchmark de la compétition Max-SAT 2014

Les résultats sur les benchmarks de la compétition Max-SAT 2014 sont montrés dans les tables 4.5, 4.6, 4.7 et 4.8 où les meilleurs résultats sont en gras. Les résultats de l'étude statistique sont donnés dans la table 4.9 et le box plot à la figure 4.4.

Bien que l'écart se soit grandement rétréci entre les 3 hyper-heuristiques, nous constatons toujours que R-HH et SCF-HH se rapprochent et dépassent CF-HH.

TABLE 4.4 – Résultats statistiques du benchmark bmc

| Methods | **Min** | **1st. Qu** | **Median** | **Mean** | **3rd Qu** | **Max** |
|---|---|---|---|---|---|---|
| **SCF-HH** | 92.62% | 96.07% | 97.50% | 96.96% | 99.13% | 99.43% |
| **CF-HH** | 91.98% | 95.57% | 96.12% | 95.75% | 96.83% | 99.12% |
| **R-HH** | 94.20% | 97.18% | 98.66% | 97.99% | 99.24% | **99.75%** |

TABLE 4.5 – Résultats des benchmarks industriels compétition Max-SAT (UNSAT) (a)

| **Benchmark** | **SCF-HH** | **CF-HH** | **R-HH** |
|---|---|---|---|
| divider-problem.dimacs_11.filtered | 88.84% | 88.95% | **89.10%** |
| divider-problem.dimacs_2.filtered | 88.65% | **88.96%** | **88.96%** |
| divider-problem.dimacs_5.filtered | 88.49% | **88.94%** | 88.77% |
| divider-problem.dimacs_8.filtered | 88.39% | **89.07%** | 88.50% |
| dividers10.dimacs.filtered | 93.33% | 91.66% | **94.68%** |
| dividers_multivec1.dimacs.filtered | 90.93% | 90.90% | **93.78%** |
| i2c-problem.dimacs_25.filtered | 88.22% | 88.24% | **88.29%** |
| i2c-problem.dimacs_26.filtered | 88.35% | 88.40% | **88.50%** |
| SM_AS_TOP_buggy1.dimacs.filtered | **97.92%** | 97.68% | 97.86% |
| SM_MAIN_MEM_buggy1.dimacs.filtered | **94.74%** | 94.31% | 94.62% |
| SM_RX_TOP.dimacs.filtered | 93.08% | **93.93%** | 92.84% |
| fpu_multivec1-problem.dimacs_14.filtered | 92.37% | 92.19% | **92.93%** |

TABLE 4.6 – Résultats des benchmarks industriels compétition Max-SAT (UNSAT) (b)

| **Benchmark** | **SCF-HH** | **CF-HH** | **R-HH** |
|---|---|---|---|
| c1_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | 84.77% | 84.68% | **84.92%** |
| c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | **84.68%** | 84.60% | 84.30% |
| c4_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | 90.53% | 90.50% | **90.56%** |
| c4_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | 90.69% | 90.64% | **90.72%** |
| c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | 85.63% | **86.23%** | 85.76% |
| c5_DD_s3_f1_e1_v2-bug-gate-0.dimacs.seq.filtered | 85.63% | **86.08%** | 85.81% |
| c6_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | **84.09%** | 82.30% | 82.57% |
| mem_ctrl1.dimacs.filtered | **95.00%** | **95.00%** | 94.61% |
| mem_ctrl2_blackbox_mc_dp-problem.dimacs_28.filtered | 92.65% | **92.91%** | 92.69% |
| mem_ctrl-problem.dimacs_27.filtered | **92.48%** | 92.34% | 92.29% |
| mrisc_mem2wire-problem.dimacs_29.filtered | **93.76%** | 93.75% | 93.69% |
| rsdecoder-debug.dimacs | 85.65% | 85.12% | **85.67%** |
| sudoku-debug.dimacs | 90.44% | 90.40% | **90.46%** |
| wb-debug.dimacs | 88.55% | 88.27% | **89.02%** |

TABLE 4.7 – Résultats des benchmarks industriels compétition Max-SAT (UNSAT) (c)

| Benchmark | SCF-HH | CF-HH | R-HH |
|---|---|---|---|
| rsdecoder1_blackbox_CSEEblock-problem.dimacs_32.filtered | **92.13%** | 92.08% | 91.57% |
| rsdecoder1_blackbox_KESblock-problem.dimacs_30.filtered | **91.86%** | 91.77% | 91.69% |
| rsdecoder2.dimacs.filtered | **88.93%** | 88.81% | 88.60% |
| rsdecoder4.dimacs.filtered | 89.41% | **89.66%** | 89.07% |
| rsdecoder5.dimacs.filtered | **89.26%** | 89.19% | 89.07% |
| rsdecoder6.dimacs.filtered | **89.24%** | 89.11% | 89.06% |
| rsdecoder_fsm2.dimacs.filtered | **89.23%** | 89.12% | 89.08% |
| rsdecoder_multivec1.dimacs.filtered | **91.92%** | 91.88% | 91.71% |
| rsdecoder_multivec1-problem.dimacs_33.filtered | **92.50%** | 92.36% | 92.47% |
| rsdecoder-problem.dimacs_31.filtered | 90.49% | **90.50%** | 89.81% |
| rsdecoder-problem.dimacs_36.filtered | **91.15%** | 90.71% | 89.60% |
| rsdecoder-problem.dimacs_37.filtered | **90.24%** | 89.91% | 89.25% |
| rsdecoder-problem.dimacs_38.filtered | **91.14%** | 90.21% | 89.58% |
| rsdecoder-problem.dimacs_39.filtered | **91.00%** | 90.06% | 89.46% |
| rsdecoder-problem.dimacs_40.filtered | **91.11%** | 90.02% | 89.45% |
| rsdecoder-problem.dimacs_41.filtered | **91.04%** | 90.07% | 89.51% |

TABLE 4.8 – Résultats des benchmarks industriels compétition Max-SAT (UNSAT) (d)

| Benchmark | SCF-HH | CF-HH | R-HH |
|---|---|---|---|
| wb1.dimacs.filtered | 95.69% | 94.18% | **96.25%** |
| wb2.dimacs.filtered | 95.61% | 94.02% | **96.13%** |
| wb_4m8s1.dimacs.filtered | **95.29%** | 94.85% | 94.88% |
| wb_4m8s3.dimacs.filtered | **95.20%** | 94.87% | 94.86% |
| wb_4m8s4.dimacs.filtered | **95.27%** | 94.97% | 94.98% |
| wb_4m8s-problem.dimacs_47.filtered | **92.48%** | 92.44% | 92.42% |
| wb_4m8s-problem.dimacs_48.filtered | **92.45%** | 92.40% | 92.39% |
| wb_4m8s-problem.dimacs_49.filtered | **92.47%** | 92.40% | 92.40% |
| wb_conmax1.dimacs.filtered | **98.25%** | 96.87% | 97.00% |
| wb_conmax3.dimacs.filtered | **98.06%** | 96.67% | 96.86% |
| wb-problem.dimacs_45.filtered | **91.86%** | 91.71% | 91.80% |
| wb-problem.dimacs_46.filtered | 91.84% | 91.64% | **92.52%** |

TABLE 4.9 – Résultats statistiques des benchmarks Industriels Compétition Max-SAT (a-b-c-d)

| Methods | Min | 1st. Qu | Median | Mean | 3rd Qu | Max |
|---|---|---|---|---|---|---|
| **SCF-HH** | **84.09%** | 89.00% | **91.12%** | **91.17%** | 92.61% | **98.25%** |
| **CF-HH** | 82.30% | **89.08%** | 90.67% | 90.90% | 92.43% | 97.68% |
| **R-HH** | 82.57% | 89.06% | 90.64% | 90.99% | **92.91%** | 97.86% |

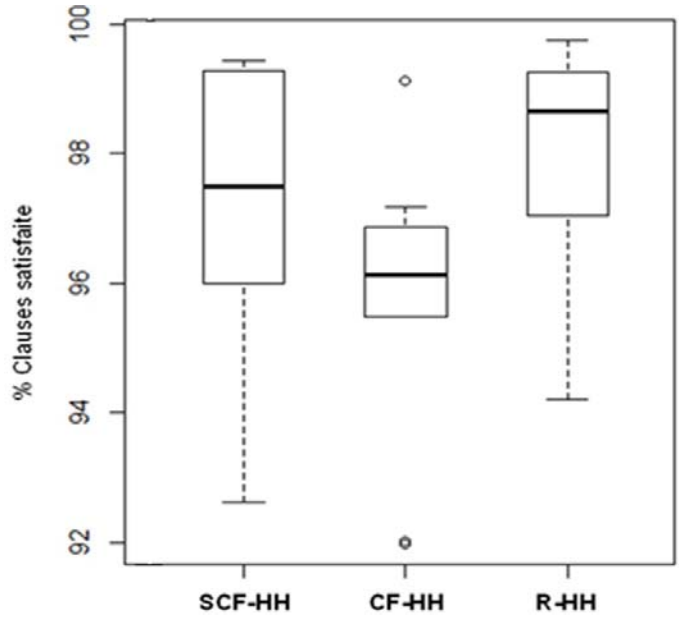



FIGURE 4.3 – Box-plot des résultats de SCF-HH, CF-HH et R-HH pour les benchmarks bmc

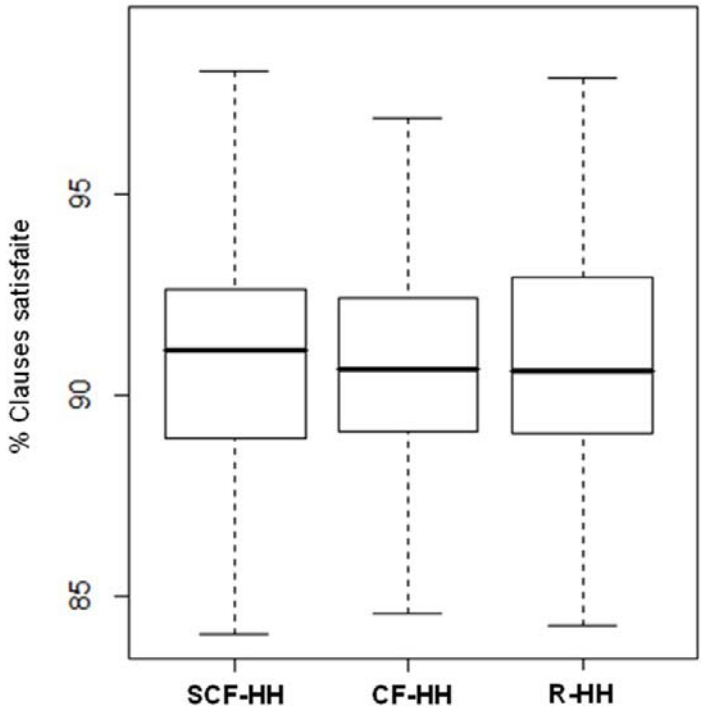



FIGURE 4.4 – Box-plot des résultats de SCF-HH, CF-HH et R-HH pour les benchmarks de la compétition Max-SAT

#### 4.2.5.3.4 Discussion des résultats

Sur l'ensemble des benchmarks (petites (Hoos), moyennes (bmc) et grande tailles (Max-SAT compétition 2014)), nous remarquons que :

- SCF-HH et R-HH se rapprochent avec un léger avantage pour SCF-HH.

- SCF-HH et R-HH donnent de meilleurs résultats que CF-HH.
- L'écart entre les trois méthodes se rétrécit au fur et à mesure que les instances deviennent plus grandes.

Donc, l'approche purement informé (CF-HH) n'a pas pu rivaliser avec l'approche aléatoire sur l'ensemble des instances, et l'approche hybride (SCF-HH) a permis de combler légèrement ce point faible.

## 4.3 Intégration du paradigme multiniveau

### 4.3.1 Architecture d'une hyper-heuristique multiniveau

Une hyper-heuristique multiniveau se compose de quatre modules :

#### 4.3.1.1 Le module de contraction

Ce module contracte le problème récursivement jusqu'à atteindre la limite désirée. Dans le cas d'une instance du problème Max-SAT, comme montré à la figure 4.5, l'algorithme fusionne des paires de variables choisis aléatoirement afin de créer ce qu'on appelle des clusters. Ces clusters sont utilisés pour définir une version plus petite et contractée du problème initial au niveau suivant. Les variables qui n'ont pas été fusionnées sont simplement copiées au niveau suivant. Les clusters du niveau courant sont alors à leurs tours regroupés deux à deux afin de définir ceux du niveau suivant, ainsi de suite. Nous avons constaté que fusionner plus de deux variables à la fois engendrait une contraction plus rapide et donc des échantillons (du problème) de qualité inférieur. Donc avec cette méthode la taille du problème est réduite à chaque itération de moitié ce qui donne une complexité temporelle logarithmique. Cette méthode de contraction est exacte car à chaque étape après l'initialisation, la solution courante peut être simplement étendu (voir algorithme d'extension dans ce qui suit) à travers les niveaux pour former une solution légitime au problème initial avec la même valeur de la fonction objectif.

#### 4.3.1.2 Le module d'initialisation

Au niveau le plus contracté, on débute le processus de résolution en générant aléatoirement une solution initiale. Les valeurs de vérité sont assignées aux clusters, donc toutes les variables composant le cluster recevront la même valeur. En effet, un cluster représente une variable aux niveaux contractés. Comme dans l'exemple de la figure 4.6 l'algorithme d'initialisation a assigné la valeur vraie pour le cluster $C_1$ et la valeur fausse pour le cluster $C_2$.

#### 4.3.1.3 Le module d'extension

Le processus d'extension étant l'inverse du processus de contraction, il divise chaque cluster du niveau courant en deux clusters dans le niveau suivant. Chacun des deux nouveaux clusters se voit assigné la même valeur que le cluster

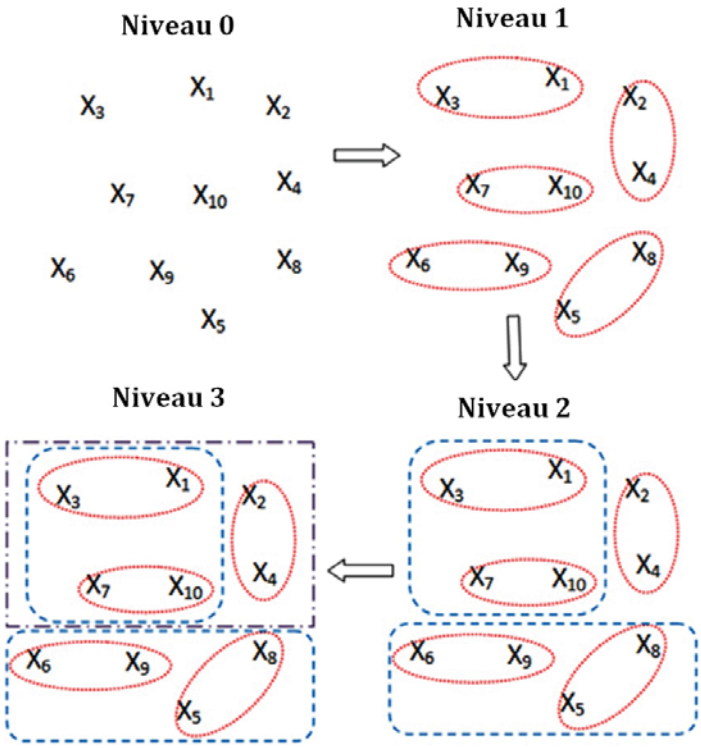



FIGURE 4.5 – Processus de contraction

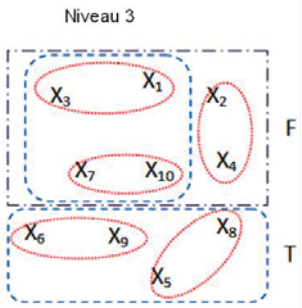



FIGURE 4.6 – Processus d'initialisation

original. La figure 4.7 illustre le passage du niveau 3 vers le niveau 2. Cette stratégie garantie que le nombre de clauses satisfaites par la solution reste le même après l'extension.

#### 4.3.1.4 Le module de raffinement

Le processus de raffinement consiste à chercher la meilleure solution correspondant à l'échantillon du problème du niveau courant. Il commence par une solution initiale passée du niveau précèdent. Le processus est déclenché à chaque niveau jusqu'à arriver au niveau 0 (contenant le problème original) voir figure 4.7. Dans notre cas le processus de raffinement est effectué par l'hyper-heuristique stochastique choice-function décrite précédemment. Dans les niveaux les plus contractés, les heuristiques de bas niveau de l'hyper-heuristique parcourent les espaces de recherche de versions plus petite, et donc plus facile à résoudre que le problème original (du niveau 0). Nous avons appliqué quelques modifica-

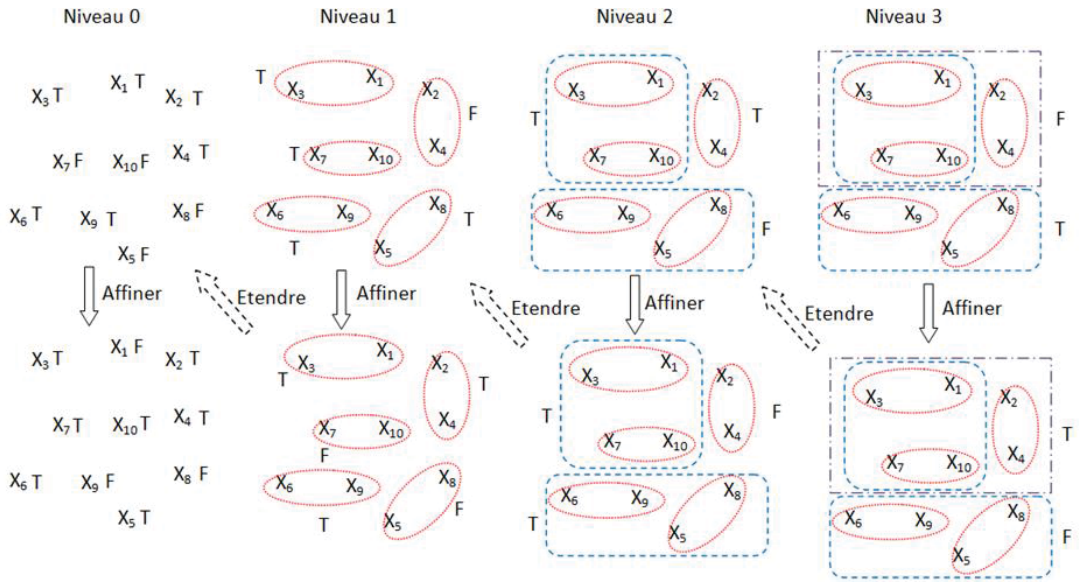



FIGURE 4.7 – Processus d'extension et de raffinement

tions à l'hyper-heuristique pour qu'elle puisse manipuler les clusters au lieu de variables.

L'algorithme 7 montre comment l'hyper-heuristique stochastique choice function a été intégrée au processus multiniveau.

**Algorithm 7** Multiniveau Stochastique Choice-Function Hyper-heuristique (ML-SCF-HH)

**Entrée(s)** Une instance Max-SAT, nombre de niveau maximum L.
**Sortie(s)** Une solution S.

1: Niveau= 0 ;
2: **// Phase de contraction**
3: **Tantque** (Niveau < L) **faire**
4: $P_{Niveau+1}$ := Contraction($P_{Niveau}$) ;
5: Niveau= Niveau+1 ;
6: **FinTantque**
7: **// Phase d'initialisation**
8: $S_L$ := Solution_Initiale($P_L$) ;
9: **// Phase d'extension et de raffinement.**
10: **Tantque** (Niveau > 0) **faire**
11: **// etendre le probleme et projeter la solution du niveau precedent**
12: $S_{Niveau-1}$ =Etendre($S_{Niveau}$) ;
13: Niveau := Niveau - 1 ;
14: **//Rafiner la solution (appeler l'Hyper-heuristique)**
15: $S_{Niveau}$ :=StochastiqueCFHH($S_{Niveau}$) ;
16: **FinTantque**
17: **Retourner** $S_{Niveau}$

TABLE 4.10 – Resultats pour les benchmarks Hoos (SAT)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| Uf250-01 | **100.00%** | 99.91% | **100.00%** | **100.00%** |
| Uf250-02 | 99.81% | **100.00%** | **100.00%** | 99.91% |
| Uf250-03 | 99.81% | 99.62% | **100.00%** | **100.00%** |
| Uf250-04 | 99.91% | 99.91% | **100.00%** | **100.00%** |
| Uf250-05 | 99.72% | 99.72% | 99.72% | **99.91%** |
| Uf250-06 | 99.53% | 99.72% | **100.00%** | **100.00%** |
| Uf250-07 | 99.72% | 99.62% | 99.81% | **99.91%** |
| Uf250-08 | 99.81% | 99.72% | **100.00%** | **100.00%** |
| Uf250-09 | 99.81% | 99.81% | 99.81% | **99.91%** |
| Uf250-010 | 99.53% | 99.81% | 99.81% | **100.00%** |
| Uf250-011 | **99.81%** | **99.81%** | 99.72% | **99.81%** |
| Uf250-012 | 99.91% | **100.00%** | **100.00%** | **100.00%** |
| Uf250-088 | 99.81% | **99.91%** | **99.91%** | **99.91%** |
| Uf250-089 | 99.81% | 99.62% | **100.00%** | 99.91% |
| Uf250-090 | 99.81% | 99.62% | 99.91% | **100.00%** |
| Uf250-091 | **100.00%** | 99.91% | **100.00%** | **100.00%** |
| Uf250-092 | 99.44% | 99.53% | **100.00%** | **100.00%** |
| Uf250-093 | **99.91%** | 99.81% | **99.91%** | **99.91%** |
| Uf250-094 | 99.91% | 99.72% | **100.00%** | **100.00%** |
| Uf250-095 | 99.81% | 99.91% | **100.00%** | **100.00%** |
| Uf250-096 | 99.72% | 99.81% | 99.91% | **100.00%** |
| Uf250-097 | 99.62% | 99.91% | 99.81% | **100.00%** |
| Uf250-098 | 99.91% | 99.91% | **100.00%** | **100.00%** |
| Uf250-099 | 99.81% | 99.91% | 99.91% | **100.00%** |
| Uf250-0100 | 99.81% | 99.72% | 99.91% | **100.00%** |

### 4.3.2 Résultats expérimentaux

Nous allons comparer l'hyper-heuristique Stochastique Choice-Function avec sa version multiniveau, ainsi que deux méthodes bien connues de l'état de l'art qui sont GSAT [Selman et al., 1992] et WalkSAT[Selman et al., 1994]. Les résultats sont montrés dans les tables 4.10, 4.12, 4.14, 4.15, 4.16 et 4.17. Les résultats de l'étude statistique sont donnés dans les tables 4.11, 4.13 et 4.18 et les box plots dans les figures 4.8, 4.9 et 4.10.

Nous constatons que WalkSAT et GSAT ont une meilleure performance par rapport aux autres hyper-heuristiques sur les benchmarks Hoos. Ceci est probablement dû à la taille des instances qui est petite. Par contre, l'avantage du paradigme multiniveau peut être constaté sur les benchmarks bmc qui sont de taille moyenne. L'efficacité de ce dernier augmente significativement avec les benchmarks de la compétition Max-SAT 2014 qui sont de très grande taille.

Les résultats expérimentaux indiquent que l'hyper-heuristique multiniveau stochastique choice function (ML-SCF-HH) est la plus robuste. Ceci est dû à deux raisons :

- L'hybridation de la méthode de sélection des heuristiques de bas niveaux,

TABLE 4.11 – Résultats statistiques des benchmarks Hoos

| Methods | Min | 1st. Qu | Median | Mean | 3rd Qu | Max |
|---|---|---|---|---|---|---|
| **ML-SCF-HH** | 99.44% | 99.72% | 99.81% | 99.79% | 99.91% | 100.00% |
| **SCF-HH** | 99.53% | 99.72% | 99.81% | 99.80% | 99.91% | 100.00% |
| **WalkSAT** | 99.72% | **99.91%** | **100.00%** | 99.93% | **100.00%** | 100.00% |
| **GSAT** | **99.81%** | **99.91%** | **100.00%** | **99.97%** | **100.00%** | 100.00% |

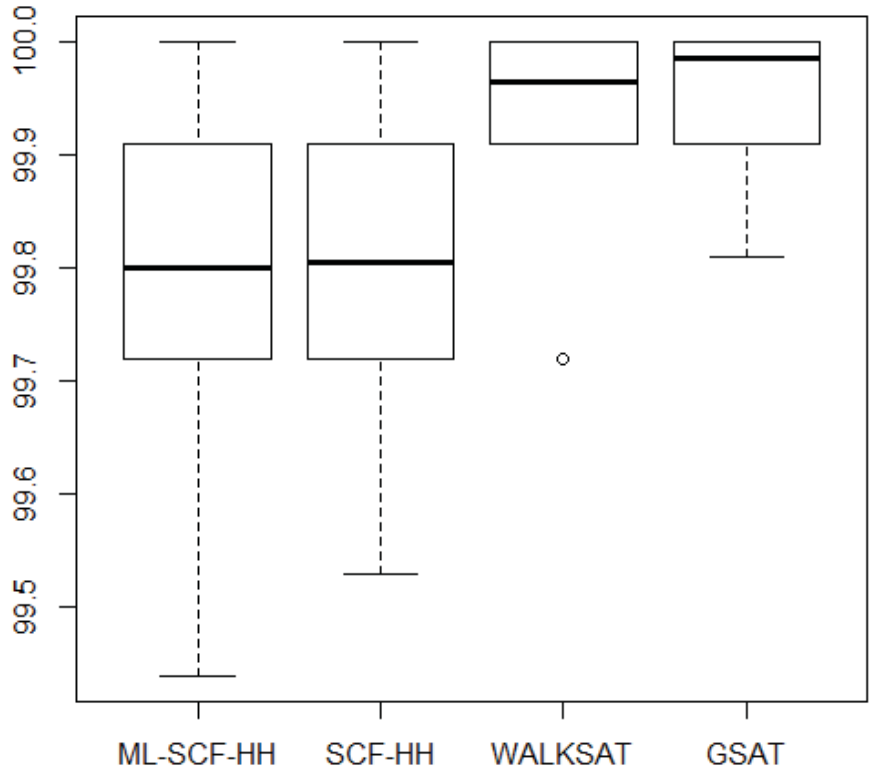



FIGURE 4.8 – Box-plot des résultats de ML-SCF-HH, SCF-HH, WalkSAT et GSAT pour les benchmarks de Hoos

qui combine :

  - la méthode choice-function qui utilise les informations récolté lors du processus de recherche afin d'améliorer la sélection.
  - La méthode aléatoire qui offre une meilleure diversification, et donc offre l'opportunité aux heuristiques de bas niveaux d'explorer des régions de l'espace de recherche qui n'auraient pas été atteintes autrement.

- Le paradigme multiniveau, qui contracte efficacement le problème afin de créer des échantillons plus petits et donc plus facile à résoudre. En flippant la valeur d'un cluster, la méthode de résolution traite plusieurs variable à la fois. Cela permet d'explorer efficacement l'espace de recherche, avec un bon équilibre entre l'intensification et la diversification. Au niveau 0

TABLE 4.12 – Résultats du benchmark bmc (SAT)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| bmc-ibm-7 | **99.38%** | 99.29% | 98.27% | 99.13% |
| bmc-ibm-5 | 99.28% | **99.43%** | 95.22% | 99.08% |
| bmc-ibm-1 | **99.64%** | 99.38% | 99.11% | 99.56% |
| bmc-ibm-13 | **99.20%** | 98.66% | 94.43% | 99.19% |
| bmc-ibm-3 | 99.09% | 98.39% | 97.67% | **99.12%** |
| bmc-ibm-11 | 97.79% | 96.60% | 93.05% | **99.03%** |
| bmc-ibm-12 | 97.23% | 96.28% | 89.93% | **99.15%** |
| bmc-galileo-8 | **99.14%** | 92.90% | 83.17% | 98.86% |
| bmc-ibm-10 | 96.83% | 96.00% | 88.77% | **98.67%** |
| bmc-galileo-9 | **99.05%** | 92.62% | 88.80% | 98.50% |

TABLE 4.13 – Résultats statistiques des benchmarks bmc

| Methods | Min | 1st. Qu | Median | Mean | 3rd Qu | Max |
|---|---|---|---|---|---|---|
| **ML-SCF-HH** | 96.83% | 98.11% | **99.11%** | 98.66% | **99.26%** | **99.64%** |
| **SCF-HH** | 92.62% | 96.07% | 97.50% | 96.95% | 99.13% | 99.43% |
| **WalkSAT** | 83.17% | 89.08% | 93.74% | 92.84% | 97.06% | 99.11% |
| **GSAT** | **98.50%** | **98.90%** | 99.10% | **99.03%** | 99.14% | 99.56% |

(l'instance d'origine), la recherche commence avec une solution initiale de très bonne qualité, ce qui permet de mieux se rapprocher de l'optimum global.

## 4.4 Conclusion

Dans ce chapitre nous avons comparé trois hyper-heuristiques : L'hyper-heuristique aléatoire (R-HH), l'hyper-heuristique choice function (CF-HH) et l'hyper-heuristique stochastique choice function (SCF-HH). Les résultats montrent que choice function, qui est une méthode basée sur l'apprentissage par renforcement, fus battu par la méthode aléatoire. L'hybridation des deux méthodes, a permis d'améliorer légèrement les résultats, mais cela reste insuffisant. L'introduction du paradigme multiniveau dans l'hyper-heuristique stochastique choice function a nettement amélioré ses performances notamment sur les benchmarks industriels Max-SAT de très grandes tailles. Ce qui confirme l'apport important du paradigme. Nous avons vu que l'hyper-heuristique stochastique choice function multiniveau a battu les autres hyper-heuristiques et les deux méthodes GSAT et WalkSAT.

Dans le chapitre suivant, nous essayons d'expliquer la faible différence trouvée entre l'hyper-heuristique aléatoire et l'hyper-heuristique choice function. Nous proposons une approche alternative afin d'essayer de palier à ce problème.

TABLE 4.14 – Résultats des benchmarks industriels de la compétition Max-SAT 2014 (UNSAT) (a)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| divider-problem.dimacs_11.filtered | **97.21%** | 88.84% | 87.18% | 85.75% |
| divider-problem.dimacs_2.filtered | **97.15%** | 88.65% | 85.20% | 85.52% |
| divider-problem.dimacs_5.filtered | **97.15%** | 88.49% | 85.24% | 85.41% |
| divider-problem.dimacs_8.filtered | **97.13%** | 88.39% | 85.02% | 85.06% |
| dividers10.dimacs.filtered | **99.02%** | 93.33% | 90.01% | 98.01% |
| dividers_multivec1.dimacs.filtered | **94.01%** | 90.93% | 85.72% | 91.42% |
| i2c-problem.dimacs_25.filtered | **97.22%** | 88.22% | 81.00% | 81.93% |
| i2c-problem.dimacs_26.filtered | **97.33%** | 88.35% | 81.19% | 82.38% |
| SM_AS_TOP_buggy1.dimacs.filtered | **98.74%** | 97.92% | 87.80% | 89.93% |
| SM_MAIN_MEM_buggy1.dimacs.filtered | **98.19%** | 94.74% | 84.34% | 84.87% |
| SM_RX_TOP.dimacs.filtered | **96.64%** | 93.08% | 86.26% | 87.67% |
| fpu_multivec1-problem.dimacs_14.filtered | **98.98%** | 92.37% | 83.74% | 85.38% |

TABLE 4.15 – Résultats des benchmarks industriels de la compétition Max-SAT 2014 (UNSAT) (b)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| c1_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | **91.57%** | 84.77% | 78.95% | 79.81% |
| c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | **91.13%** | 84.68% | 79.79% | 81.22% |
| c4_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | **90.76%** | 90.53% | 76.57% | 77.04% |
| c4_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered | **90.75%** | 90.69% | 77.26% | 77.60% |
| c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | **98.62%** | 85.63% | 79.91% | 83.46% |
| c5_DD_s3_f1_e1_v2-bug-gate-0.dimacs.seq.filtered | **98.63%** | 85.63% | 82.80% | 83.43% |
| c6_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered | **91.16%** | 84.09% | 79.55% | 80.64% |
| mem_ctrl1.dimacs.filtered | **97.87%** | 95.00% | 83.24% | 84.10% |
| mem_ctrl2_blackbox_mc_dp-problem.dimacs_28.filtered | **98.53%** | 92.65% | 82.63% | 83.08% |
| mem_ctrl-problem.dimacs_27.filtered | **98.38%** | 92.48% | 82.50% | 82.71% |
| mrisc_mem2wire-problem.dimacs_29.filtered | **98.36%** | 93.76% | 82.68% | 83.26% |
| rsdecoder-debug.dimacs | **91.17%** | 85.65% | 82.44% | 83.81% |
| sudoku-debug.dimacs | **92.78%** | 90.44% | 81.15% | 82.47% |
| wb-debug.dimacs | **92.59%** | 88.55% | 80.22% | 86.82% |

TABLE 4.16 – Résultats des benchmarks industriels de la compétition Max-SAT 2014 (UNSAT) (c)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| rsdecoder1_blackbox_CSEEblock-problem.dimacs_32.filtered | **94.88%** | 92.13% | 83.34% | 85.23% |
| rsdecoder1_blackbox_KESblock-problem.dimacs_30.filtered | **95.79%** | 91.86% | 83.22% | 83.70% |
| rsdecoder2.dimacs.filtered | **92.39%** | 88.93% | 84.92% | 86.18% |
| rsdecoder4.dimacs.filtered | **92.46%** | 89.41% | 86.31% | 87.62% |
| rsdecoder5.dimacs.filtered | **92.45%** | 89.26% | 86.22% | 88.12% |
| rsdecoder6.dimacs.filtered | **92.43%** | 89.24% | 86.13% | 87.98% |
| rsdecoder_fsm2.dimacs.filtered | **92.08%** | 89.23% | 85.72% | 87.61% |
| rsdecoder_multivec1.dimacs.filtered | **93.24%** | 91.92% | 85.60% | 86.62% |
| rsdecoder_multivec1-problem.dimacs_33.filtered | **94.49%** | 92.50% | 83.24% | 84.01% |
| rsdecoder-problem.dimacs_31.filtered | **94.46%** | 90.49% | 82.15% | 82.64% |
| rsdecoder-problem.dimacs_36.filtered | **94.46%** | 91.15% | 82.11% | 82.58% |
| rsdecoder-problem.dimacs_37.filtered | **98.44%** | 90.24% | 82.20% | 82.54% |
| rsdecoder-problem.dimacs_38.filtered | **98.41%** | 91.14% | 82.18% | 82.60% |
| rsdecoder-problem.dimacs_39.filtered | **94.44%** | 91.00% | 82.18% | 82.60% |
| rsdecoder-problem.dimacs_40.filtered | **94.44%** | 91.11% | 82.17% | 82.73% |
| rsdecoder-problem.dimacs_41.filtered | **94.45%** | 91.04% | 82.19% | 82.61% |

TABLE 4.17 – Résultats des benchmarks industriels de la compétition Max-SAT 2014 (UNSAT) (d)

| Benchmark | ML-SCF-HH | SCF-HH | WalkSAT | GSAT |
|---|---|---|---|---|
| wb1.dimacs.filtered | **98.51%** | 95.69% | 80.36% | 98.11% |
| wb2.dimacs.filtered | **98.44%** | 95.61% | 83.52% | 97.75% |
| wb_4m8s1.dimacs.filtered | **97.34%** | 95.29% | 84.13% | 84.82% |
| wb_4m8s3.dimacs.filtered | **97.33%** | 95.20% | 84.04% | 84.70% |
| wb_4m8s4.dimacs.filtered | **97.33%** | 95.27% | 84.08% | 84.78% |
| wb_4m8s-problem.dimacs_47.filtered | **96.96%** | 92.48% | 81.78% | 81.93% |
| wb_4m8s-problem.dimacs_48.filtered | **96.95%** | 92.45% | 81.83% | 81.97% |
| wb_4m8s-problem.dimacs_49.filtered | **97.20%** | 92.47% | 81.78% | 81.93% |
| wb_conmax1.dimacs.filtered | **98.54%** | 98.25% | 85.22% | 86.79% |
| wb_conmax3.dimacs.filtered | **98.55%** | 98.06% | 85.72% | 86.72% |
| wb-problem.dimacs_45.filtered | **95.15%** | 91.86% | 80.61% | 81.95% |
| wb-problem.dimacs_46.filtered | **96.06%** | 91.84% | 80.11% | 82.14% |

TABLE 4.18 – Résultats statistique des benchmarks de la compétition Max-SAT 2014

| Methods | Min | 1st. Qu | Median | Mean | 3rd Qu | Max |
|---|---|---|---|---|---|---|
| **ML-SCF-HH** | **90.75%** | **93.43%** | **96.95%** | **95.72%** | **98.32%** | **99.02%** |
| **SCF-HH** | 84.09% | 89.00% | 91.12% | 91.17% | 92.61% | 98.25% |
| **WalkSAT** | 76.57% | 81.78% | 82.74% | 83.06% | 85.16% | 90.01% |
| **GSAT** | 77.04% | 82.55% | 83.91% | 84.85% | 86.51% | 98.11% |

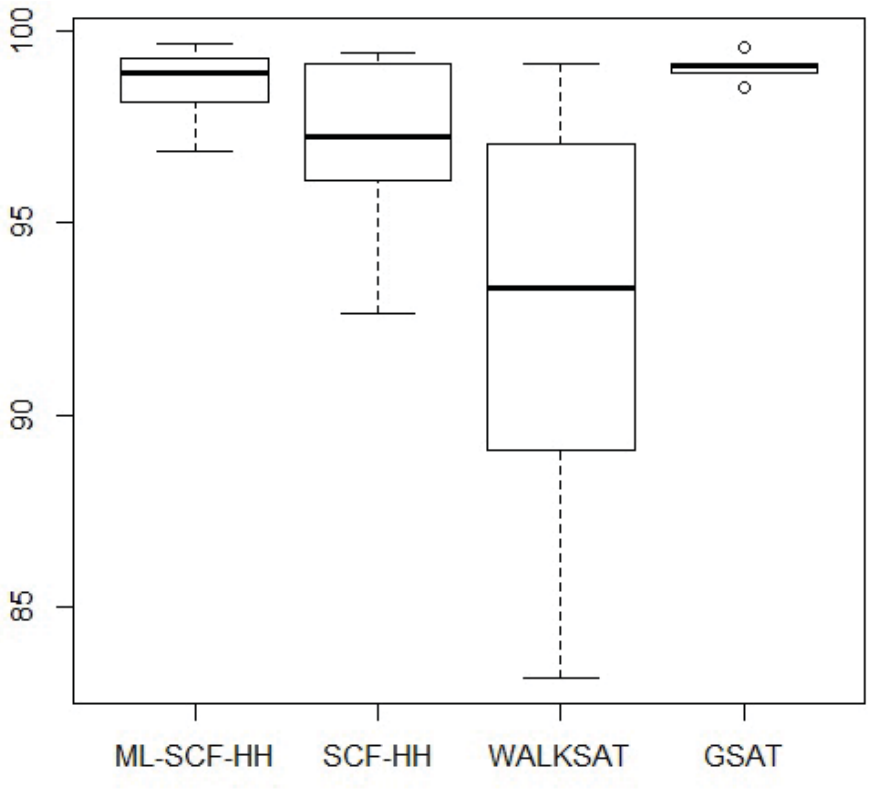



FIGURE 4.9 – Box-plot des résultats de ML-SCF-HH, SCF-HH, WalkSAT et GSAT pour les benchmarks bmc

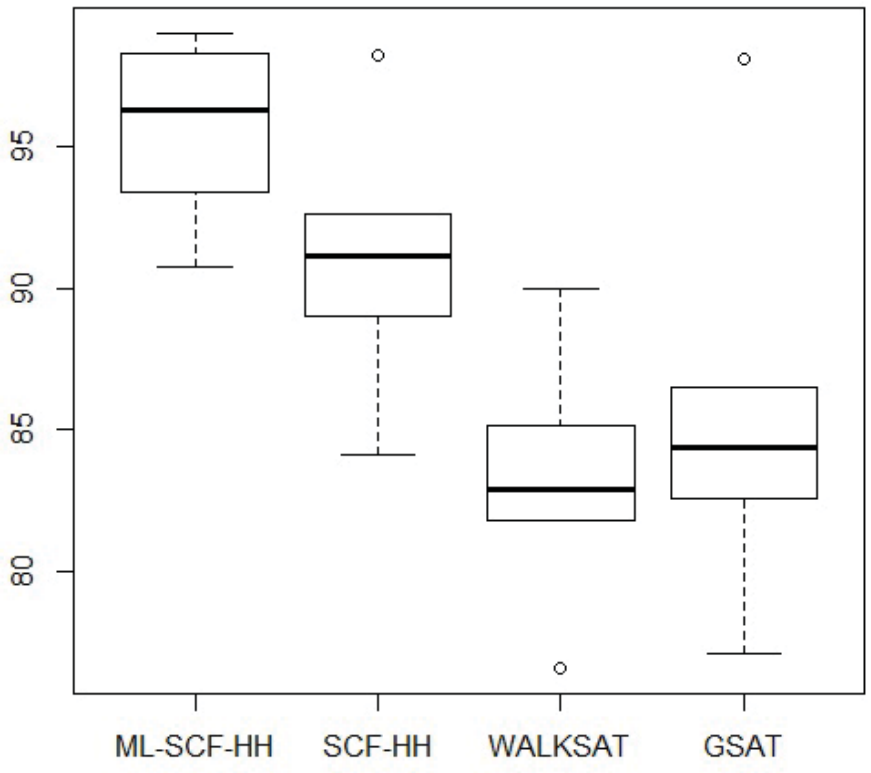



FIGURE 4.10 – Box-plot des résultats de ML-SCF-HH, SCF-HH, WalkSAT et GSAT pour les benchmarks industriels de la compétition Max-SAT

# Chapitre 5

# Une nouvelle approche Hyper-heuristique probabiliste sur le problème Max-SAT

## 5.1 Introduction

Dans le chapitre précèdent, nous avons constaté que choice function, qui est une méthode d'apprentissage par renforcement additive, n'arrivait pas à battre la méthode de sélection aléatoire. Dans ce qui suit, nous allons expliquer l'inconvénient de ce type d'apprentissage par renforcement, et proposer une alternative afin de palier à ce problème.

## 5.2 Inconvénient des hyper-heuristiques additives et la méthode Thompson Sampling

Une hyper-heuristique utilisant le mécanisme d'apprentissage par renforcement, choisis l'heuristique la plus appropriée à chaque itération, en utilisant le principe d'*essais* et d'*erreurs*. A chaque heuristique de bas niveau on associe un poids, initialement de même valeur. Un module d'apprentissage par renforcement gère la mise à jour de ces poids. Le schéma d'*apprentissage additif*, qui est basé sur l'addition de poids, est le plus utilisé. Si l'heuristique de bas niveau sélectionnée améliore la solution, son poids augmente d'une certaine valeur, dans le cas contraire le poids est diminué. Un exemple d'une hyper-heuristique utilisant ce mécanisme est la choice function.

Parmi les travaux les plus récents utilisant choice function on peut citer : [Pour et al., 2018] où l'hyper-heuristique a été utilisée sur le problème d'allocation des tâches de maintenance pour les chemins de fer danois. Dans [Choong et al., 2017, Choong et al., 2019] un algorithme de colonie d'abeilles artificielle est combiné dans la choice function pour le problème du voyageur de commerce. Dans [Din et al., 2017], une hyper-heuristique avec une choice function sans paramètres a été appliquée sur la génération automatique de tests de comparaison par paires. Dans [Drake et al., 2015] une choice function modifiée a été proposée pour le problème du sac à dos multidimensionnel. Dans

[Chifu et al., 2018] une hyper-heuristique constructive est utilisée pour générer automatiquement des menues sains personnalisés.

Dans [Alanazi and Lehre, 2016], les limites du mécanisme d'apprentissage par renforcement additif sont démontrées. Les auteurs ont prouvé théoriquement que si la probabilité de succès des heuristiques de bas niveau est inférieure à $\frac{1}{2}$, alors l'hyper-heuristique aura la même performance qu'un mécanisme aléatoire. Dans leurs analyses expérimentales, une hyper-heuristique additive a été implémentée en utilisant la plateforme HyFlex, puis a été appliquée au problème *bin-packing* et au problème flow-shop. Les résultats montrent que la probabilité de succès des heuristiques de bas niveau est très inférieure à $\frac{1}{2}$, et par conséquence, l'hyper-heuristique avec apprentissage par renforcement additif et l'hyper-heuristique aléatoire ont eu asymptotiquement le même comportement. Cela montre que le mécanisme d'apprentissage par renforcement additif n'est pas nécessairement capable de distinguer les performances des différentes heuristiques de bas niveau. En d'autres termes, ce mécanisme ne s'adapte pas afin de suivre dynamiquement le changement des probabilités de succès des heuristiques de bas niveau.

Puisque ce mécanisme est inutile dans ces cas-là [Alanazi, 2016] propose l'utilisation d'une approche de sélection basée probabilités appelée **Thompson Sampling**.

## 5.3 L'hyper-heuristique Thompson Sampling

En 1933, Thompson a introduit un nouveau mécanisme d'apprentissage par renforcement pour le problème du bandit manchot (*the multi-armed bandit problem*) appelée **Thompson Sampling** [Thompson, 1933]. Bien que cette technique ait été absente de la littérature de l'intelligence artificielle, récemment, elle a suscité un intérêt considérable. Plusieurs études ont démontrées empiriquement l'efficacité de Thompson Sampling [Agrawal and Goyal, 2013, Granmo, 2010, May and Leslie, 2011, Scott, 2010]. Elle a été également appliquée avec succès sur plusieurs problèmes réels [Chapelle and Li, 2011, Graepel et al., 2010, Tang et al., 2013].

Comme le montre algorithme 8, Thompson Sampling est un mécanisme de renforcement qui utilise les probabilités pour prédire l'heuristique de bas niveau la plus adaptée à chaque étape de la recherche. Il utilise aussi une fenêtre temporelle glissante (*sliding time window*) afin d'adapter son comportement aux récentes observations sur les performances des heuristiques de bas niveau. Cette fenêtre permet de délaisser les anciennes observations (qui sont potentiellement obsolètes). Dans ce cas, les heuristiques de bas niveau sont divisées en deux groupes : MU (heuristiques de mutation) et LS (heuristiques de simple recherches locales). L'hyper-heuristique choisit une heuristique à partir de l'ensemble MU puis elle en choisit une deuxième à partir de l'ensemble LS. A chaque heuristique de bas niveau $i$, on attribue une beta distribution avec deux paramètres $\alpha_i^{(t)}$ et $\beta_i^{(t)}$ qui représentent le nombre de succès et le nombre d'Echecs observés dans l'intervalle de la fenêtre temporelle à l'itération $t$. Ces paramètres sont initialisés à 1, et sont mis à jour durant le processus de recherche. Dans le cas d'un succès, ce qui veut dire que l'heuristique de bas niveau

a amélioré la meilleure solution trouvée par rapport à la fonction objectif, $\alpha_i^{(t)}$ est incrémenté. Dans le cas contraire (un échec) $\beta_i^{(t)}$ est incrémenté. Dans le but de sélectionner la prochaine heuristique de bas niveau à être appelée, une variable aléatoire $U_i^{(t)}$ appelée score d'utilité (*utility score*) est extraite à partir de la beta distribution de chaque heuristique de bas niveau. Celle avec le plus grand score d'utilité est sélectionnée. L'hyper-heuristique conserve les observations (les succès et les échecs de chaque heuristique de bas niveau) des **w** dernières itérations.

L'hyper-heuristique Thompson Sampling sélectionne, à chaque itération, l'heuristique qui va, potentiellement, améliorer la solution candidate, sans prendre en considération la synergie entre les heuristiques de bas niveau. Comme nous l'avons mentionné précédemment, les hyper-heuristiques tentent de compenser les faiblesses de certaines heuristiques de bas niveau par les forces d'autres heuristiques. Cependant, l'hyper-heuristique Thompson Sampling ne prend pas cela en considération.

Nous proposons dans ce qui suit, notre contribution qui consiste en une nouvelle approche qui rajoute l'aspect synergie à l'hyper-heuristique Thompson Sampling.

## 5.4 L'hyper-heuristique Synergie Thompson Sampling

L'hyper-heuristique Synergie Thompson Sampling s'appuie sur une approche probabiliste afin de gérer la sélection des heuristiques de bas niveau. Elle utilise l'historique des performances des heuristiques de bas niveau pour mettre à jour le mécanisme d'apprentissage qui se base sur la loi de probabilité Beta. La loi de probabilité Beta est généralement utilisée pour modéliser l'incertain de la probabilité de réussite d'une expérimentation. C'est une distribution de probabilité continue définie dans l'intervalle $[0, 1]$, et possède deux paramètres $\alpha$ et $\beta$ qui contrôlent la forme de la courbe de la distribution. Par exemple, pour prédire le succès d'une expérimentation, une variable aléatoire $x$ est extraite de sa distribution Beta, où le paramètre $\alpha$ représente le nombre de succès, et $\beta$ représente le nombre d'échecs. Dans notre cas, la distribution Beta est utilisée afin de mesurer la synergie entre les heuristiques de bas niveau $i$ et $j$, en suivant le comportement de la séquence dans un certain intervalle de temps durant tout le processus de résolution. Donc, à chaque combinaison d'heuristiques $(i, j)$, une distribution Beta$(\alpha_{ij}^{(t)}, \beta_{ij}^{(t)})$ est affectée. A l'itération $t$, le paramètre $\alpha_{ij}^{(t)}$ est incrémenté dans le cas d'un succès, dans le cas d'un échec, le paramètre $\beta_{ij}^{(t)}$ est incrémenté. Un succès veut dire que la séquence des heuristiques $i$ et $j$ de bas niveau à améliorer la solution initialement passée à $i$. Dans ce cas, la distribution Beta capture (approximativement) le comportement de la séquence $(i, j)$. Donc, si dans l'itération courante l'heuristique $i$ a été appelée, on peut prédire la probabilité de succès de l'heuristique j dans la prochaine itération, en calculant une variable aléatoire $U_{ij}^{(t)}$ appelée score d'utilité (*utility score*). Les couples $(\alpha_{ij}^{(t)}, \beta_{ij}^{(t)})$ sont sauvegardés dans une matrice $(L * L)$, où $L$ est le nombre d'heuristique de bas niveau. Dans notre cas, nous n'avons pas

**Algorithm 8** L'hyper-heuristique Thompson Sampling (TS-HH).

**Entrée(s)** Une instance Max-SAT, Un ensemble $H$ de $m$ heuristiques de bas niveau (un sous-ensemble "$MU$" contenant des heuristiques de mutation et d'un sous-ensemble "$LS$" d'heuristique à recherché locale), le paramètre $maxiter$, la fenêtre temporelle glissante $w \in \mathbb{N}$.

**Sortie(s)** Une solution $S$.

1: Generer une solution initiale $S$.
2: $F :=$ Evaluation de la solution $S$.
3: $t := 0$;
4: $\forall i \in [m]$, initialiser les paramètres $\alpha_i^{(0)} := 1$, et $\beta_i^{(0)} := 1$
5: $\forall i \in [m]$, soit $U_i^{(0)}$ le score d'utilité de l'heuristique de bas niveau $i$
6: **Tantque** $(t < maxiter)$ **faire**
7: **//Heuristique de sélection**
8: $\forall i \in [m]$, échantillonner $U_i^{(t)}$ à partir de Beta$(\alpha_i^{(t)}, \beta_i^{(t)})$
9: $h_i :=$ Une heuristique dans $MU$ ayant le plus grand score d'utilité $U_i^{(t)}$ ;
10: $h_j :=$ Une heuristique dans $LS$ ayant le plus grand score d'utilité $U_j^{(t)}$ ;
11: Appliquer l'heuristique $h_i$ sur $S$ pour obtenir $S'$ avec une qualité $F'$
12: Appliquer l'heuristique $h_j$ sur $S'$ pour obtenir $S''$ avec une qualité $F''$
13: **Si** $(F'' > F)$ **Alors**
14: ${\alpha_i}^{(t+1)} := {\alpha_i}^{(t)} + 1$
15: ${\alpha_j}^{(t+1)} := {\alpha_j}^{(t)} + 1$
16: **Sinon**
17: ${\beta_i}^{(t+1)} := {\beta_i}^{(t)} + 1$
18: ${\beta_j}^{(t+1)} := {\beta_j}^{(t)} + 1$
19: **Finsi**
20: **Si** $(t \geq w)$ **Alors**
21: $\forall i \in [m]$ si à l'itération $(t - w)$, $h_i$ fut appelé et avait amélioré la solution, alors ${\alpha_i}^{(t+1)} := {\alpha_i}^{(t+1)} - 1$
22: $\forall i \in [m]$ si à l'itération $(t - w)$, $h_i$ fut appelé et n'avait pas amélioré la solution, alors ${\beta_i}^{(t+1)} := {\beta_i}^{(t+1)} - 1$
23: **Finsi**
24: **Si** $(f(S'') \geq f(S))$ **Alors**
25: $S := S''; F := F''$ ;
26: **Finsi**
27: **FinTantque**
28: **Retourner** la meilleure solution.

besoin de séparer les heuristiques de bas niveau en deux groupes, comme il a été fait dans l'algorithme précédent, car nous allons utiliser des méthodes de l'état de l'art de Max-SAT au lieu de simples mutations et recherches locales.

Comme il est montré dans l'algorithme 9, le processus de sélection, à chaque itération, commence par choisir une heuristique $i$ aléatoirement. Ensuite, afin de sélectionner la prochaine heuristique à être appelée, des scores d'utilité $U_{ij}^{(t)}$ sont calculés à partir des distributions Beta de chaque couple $(i, j)$, $\forall j \in [1, L]$. L'heuristique $j$ ayant le score maximum sera sélectionnée. Après la fin de l'exécution de $j$, les paramètres $\alpha_{ij}^{(t+1)}$ et $\beta_{ij}^{(t+1)}$ sont mis à jour : si l'heuristique $j$ a amélioré la solution, alors $\alpha_{ij}^{(t+1)}$ est incrémenté, sinon $\beta_{ij}^{(t+1)}$ est incrémenté. Par la suite, l'heuristique $j$ devient la nouvelle heuristique $i$ pour l'itération $t+1$ ainsi de suite. Ce mécanisme permet de chercher la meilleure séquence de toutes les heuristiques de bas niveau à chaque étape de la recherche.

Dans notre cas, toutes les solutions retournées par les heuristiques de bas niveau sont acceptées. Une **fenêtre temporelle glissante** (*sliding time window*) est utilisée afin de ne prendre en considération que les récentes observations sur les performances des heuristiques de bas niveau. Cette fenêtre est de taille w et est gérée suivant le principe FIFO. Le paramètre **w** est très important et dois être fixé correctement, si **w** est trop grand, alors des informations obsolètes seront pris en compte dans le calcul, sinon s'il est trop petit, les observations gardées peuvent ne pas être suffisantes pour bien représenter le comportement des séquences des heuristiques de bas niveau.

## 5.5 Composantes des hyper-heuristiques

Toutes les hyper-heuristiques qui ont été implémenté dans cette partie, ont les composantes suivant :

### 5.5.1 Module d'acceptation de solution

Dans notre cas nous acceptons toutes les solutions retournés par les heuristiques de bas niveau.

### 5.5.2 Représentation d'une solution

Comme dans les parties précédentes, une solution est représentée par un vecteur binaire $X$ de taille $n, n$ étant le nombre de variables de l'instance. Chaque case du vecteur $X_i$ peut recevoir la valeur 0 (pour faux) ou bien 1 (pour vrai).

### 5.5.3 Fonction objectif

Comme dans les parties précédentes, une solution $X$ est évaluée en comptant le nombre de clauses insatisfaites dans l'instance. La solution optimale est celle qui obtient le nombre minimum. La fonction objectif $f$ à minimiser est définie

comme suit :

$$f(X) = \sum_{i=0}^{n} C_i \quad \text{où la clause} \quad C_i = \begin{cases} 1 & \text{si elle est satisfaite} \\ \\ 0 & \text{sinon.} \end{cases}$$

### 5.5.4 Les heuristiques de bas niveau

Nous avons choisi six heuristiques de bas niveau parmi les plus connu de l'état de l'art de Max-SAT.

- L'heuristique $h_1$ (**GSAT**) : Cette méthode choisie de flipper la variable avec le plus grand *num_improve* (le nombre de clause satisfaite – le nombre de clause insatisfaite si la variable est flippée). Dans le cas où il existe plusieurs variables avec le plus grand *num_improve*, on en choisi une aléatoirement.

- L'heuristique $h_2$ (**HSAT**) : Elle est similaire à GSAT, cependant s'il existe plusieurs variables avec le plus grand *num_improve*, on en choisi celle qui a le plus grand âge (l'âge d'une variable représente le nombre d'itération écouler depuis la dernière fois qu'elle a été flippée).

- L'heuristique $h_3$ (**SLS**) : C'est la recherche locale stochastique qui explore le voisinage d'une solution candidate, selon trois stratégies comme décrit dans le chapitre précédent.

- L'heuristique $h_4$ (**WalkSAT**) : Cet algorithme commence par choisir aléatoirement une clause insatisfaite. Si elle contient une variable ayant un gain négatif égal à zéro, alors cette variable est flippée (le gain négatif étant le nombre de clause insatisfaite si la variable est flippée). Si aucune variable ne correspond à ce critère, alors il existe deux options selon une probabilité **walk** :

  - On sélectionne la variable ayant le gain négatif minimum.
  - On sélectionne une variable aléatoirement dans la clause.

- L'heuristique $h_5$ (**Novelty**) : Dans Novelty, après avoir choisi une clause insatisfaite, la variable qui sera flippée sera sélectionnée comme suit :

  - Si la variable ayant le plus grand *num_improve* n'a pas l'âge minimum parmi les autres variables de la clause, alors elle sera sélectionnée.
  - Sinon, elle est sélectionnée avec une probabilité $1 - p$.
  - Sinon la variable avec le second plus grand *num_improve* sera sélectionnée.

- L'heuristique $h_6$ (**VNS**) : C'est une métaheuristique qui explore plusieurs voisinages pour une meilleure diversification, et utilise une recherche locale pour l'intensification. VNS commence par définir un ensemble de structures voisinages $N_1, N_2, \ldots, N_k$ qui seront explorés durant le processus de recherche. En commençant par une solution initiale, VNS appelle

une recherche locale afin d'explorer le premier voisinage N1 de la solution. Si la solution est améliorée, alors N1 est exploré encore. Sinon, VNS passe à la structure de voisinage suivante N2, ainsi de suite.

**Algorithm 9** Hyper-heuristique Synergie Thompson Sampling (SyTS-HH).

**Entrée(s)** Une instance Max-SAT, Un ensemble $H$ de $m$ heuristiques de bas niveau, le paramètre $maxiter$, la fenêtre temporelle glissante $w \in \mathbb{N}$.
**Sortie(s)** Une solution $S$.

Generer une solution initiale $S$.
$F :=$ Evaluation de la solution $S$.
$t := 0$;
$\forall i, j \in [m]$, initialiser les paramètres $\alpha_{ij}^{(0)} := 1$, et $\beta_{ij}^{(0)} := 1$
$\forall i, j \in [m]$, soit $U_{ij}^{(0)}$ le score d'utilité de la séquence d'heuristique de bas niveau $i, j$
$h_i :=$ Une heuristique de bas niveau aléatoirement dans $H$
Appliquer $h_i$ sur $S$, et mettre a jour la solution $S'$ avec une qualité $(F')$ .
**Tantque** $(t < maxiter)$ **faire**
    **//Heuristique de sélection**
    $\forall j \in [m]$, échantillonner $U_{ij}^{(t)}$ à partir de Beta$(\alpha_{ij}^{(t)}, \beta_{ij}^{(t)})$
    $h_j :=$ Une heuristique dans $H$ ayant le plus grand score d'utilité $U_{ij}^{(t)}$ ;
    Appliquer l'heuristique $h_j$ sur $S'$ pour obtenir $S''$ avec une qualité $F''$
    **Si** $(F'' > F)$ **Alors**
        $\alpha_{ij}^{(t+1)} := \alpha_{ij}^{(t)} + 1$
    **Sinon**
        $\beta_{ij}^{(t+1)} := \beta_{ij}^{(t)} + 1$
    **Finsi**
    **Si** $(t \geq w)$ **Alors**
        $\forall j \in [m]$ si à l'itération $(t - w)$, $h_j$ fut appelé après l'heuristique $h_i$ et avait amélioré la solution, alors $\alpha_{ij}^{(t+1)} := \alpha_{ij}^{(t+1)} - 1$
        $\forall j \in [m]$ si à l'itération $(t-w)$, $h_j$ fut appelé après l'heuristique $h_i$ et n'avait pas amélioré la solution, alors $\beta_{ij}^{(t+1)} := \beta_{ij}^{(t+1)} - 1$
    **Finsi**
    **//Methode d'acceptation : Toute les solutions (All moves)**
    $S := S'; F := F'$ ; $S' := S''; F' := F''$ ;
    **Si** $(F'' \leq F_{best})$ **Alors**
        $S_{best} := S''; F_{best} := F''$ ;
    **Finsi**
    $i := j$
**FinTantque**
**Retourner** $S_{best}$.

## 5.6 Résultats expérimentaux

Nous avons implémenté 5 hyper-heuristiques pour le problème Max-SAT :

- **R-HH** : pour l'hyper-heuristique aléatoire.

- **CF-HH** : pour l'hyper-heuristique choice function, dans notre cas nous avons utilisé la version la plus récente de choice function qui, contrairement à la première version, n'a plus de paramètres empiriques. Elle s'appuie sur la même formule de la version précédente, sauf que, les paramètres $\alpha$ et $\beta$ ont été remplacés par $\phi$ :

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

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

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

$$\forall i, score(h_i) = \phi g_1(h_i) + \phi g_2(h_{ID}, h_i) + \delta g_3(h_i)$$

Où :

$$\phi_t = \begin{cases} 0.99 & \text{si amélioration} \\ \max(\phi_{t-1} - 0.01, 0.01) & \text{sinon.} \end{cases}$$

Et :

$$\delta_t = 1 - \phi_t$$

- **SCF-HH** : pour l'hyper-heuristique Stochastique choice function, avec la nouvelle formule de choice function.
- **TS-HH** : pour l'hyper-heuristique Thompson Sampling.
- **SyTS-HH** : pour l'hyper-heuristique Synergie Thompson Sampling.

En prenant en compte la nature non-déterministe de ces méthodes, nous avons effectué dix tests sur chaque instance pour chaque méthode. Une étude empirique (comme dans le chapitre précédent) a été effectuée pour fixer les valeurs des paramètres.

Pour TS-HH et SyTS-HH, il n'y a qu'un seul paramètre. Il s'agit de la taille de la fenêtre glissante **w**, qui a été fixée à 30.

Pour la SCF-HH, la probabilité de marche (**wp**) est fixée à 0.3.

Concernant les heuristiques de bas niveau :

- $h_3$ (SLS) : probabilités de marche **walk1**=0.3 et **walk2**=0.6
- $h_4$ (Walksat) : probabilité de marche **walk**= 0.3
- $h_5$ (Novelty) : la probabilité de marche **walkn**= 0.4
- $h_6$ (VNS) : nombre de voisinages **k**=10

Table 5.1 – ML-TS-HH Vs TS-HH

| Benchmark | Variables | Clauses | ML-TS-HH HH | TS-HH |
|---|---|---|---|---|
| c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq | 400085 | 1121810 | **2553** | 5754 |
| i2c-problem.dimacs_25 | 521672 | 1581471 | **2091** | 4726 |
| rsdecoder1_blackbox_KESblock-problem.dimacs_30 | 707330 | 1106376 | **2323** | 5123 |
| mrisc_mem2wire-problem.dimacs_29 | 844900 | 2905976 | **3951** | 6978 |
| mem_ctrl1.dimacs | 1128648 | 4422185 | **1982** | 4145 |
| rsdecoder-problem.dimacs_36 | 1220616 | 3938467 | **7737** | 10642 |
| sudoku-debug.dimacs | 1304121 | 1554820 | **967** | 3023 |
| rsdecoder-problem.dimacs_37 | 1513544 | 4909231 | **4060** | 7321 |
| mem_ctrl2_blackbox_mc_dp-problem.dimacs_28 | 1974822 | 6795573 | **3052** | 8740 |
| mem_ctrl-problem.dimacs_27 | 4426323 | 15983633 | **28805** | 39654 |

### 5.6.1 TS-HH vs ML-TS-HH

L'efficacité du paradigme multiniveau a été prouvée dans la partie précédente. Pour voir l'impact de ce paradigme sur l'hyper-heuristique Thompson Sampling, nous l'avons testé sur les 10 benchmarks industriel les plus larges de la compétition Max-SAT 2014. Les résultats sont indiqués dans la table 5.1.

La figure 5.1, donne une représentation graphique des résultats où nous pouvons voir clairement la supériorité de ML-TS-HH par rapport à TS-HH. Cela confirme encore une fois l'efficacité du paradigme multiniveau sur les grandes instances.

Par la suite, nous allons appliquer le paradigme multiniveau sur les autres hyper-heuristiques.

### 5.6.2 Comparaison entre les 5 hyper-heuristiques

Comme nous nous sommes intéressés aux instances industrielles très larges des benchmarks de la compétition Max-SAT 2014, nous avons appliqué le paradigme multiniveau au 5 hyper-heuristiques pour booster la recherche.

Les résultats obtenus sont montrés dans les tables 5.2 et 5.3, où le meilleur résultat entre les 5 hyper-heuristiques est écrit en gras. Ils sont exprimés en nombre de clauses insatisfaites. La dernière colonne des tables 5.2 et 5.3 représente le meilleur résultat trouvé par les méthodes incomplètes lors de la compétition Max-SAT 2014.

Pour une meilleure interprétation des résultats, la table 5.4 donne quelque mesure statistique calculées à partir des pourcentages de satisfaction de clauses (nombre de clauses / le nombre totale des clauses de l'instance). Pour chaque méthode, nous donnons le minimum (*min*), le maximum (*max*), la moyenne (*mean*), la médiane (*median*), le premier quartile (*1er Qu*) et le troisième quartile (*3eme Qu*). La figure 5.2 illustre le diagramme box plot afin de mieux visualiser la distribution des valeurs des taux de satisfactions pour chaque méthode.

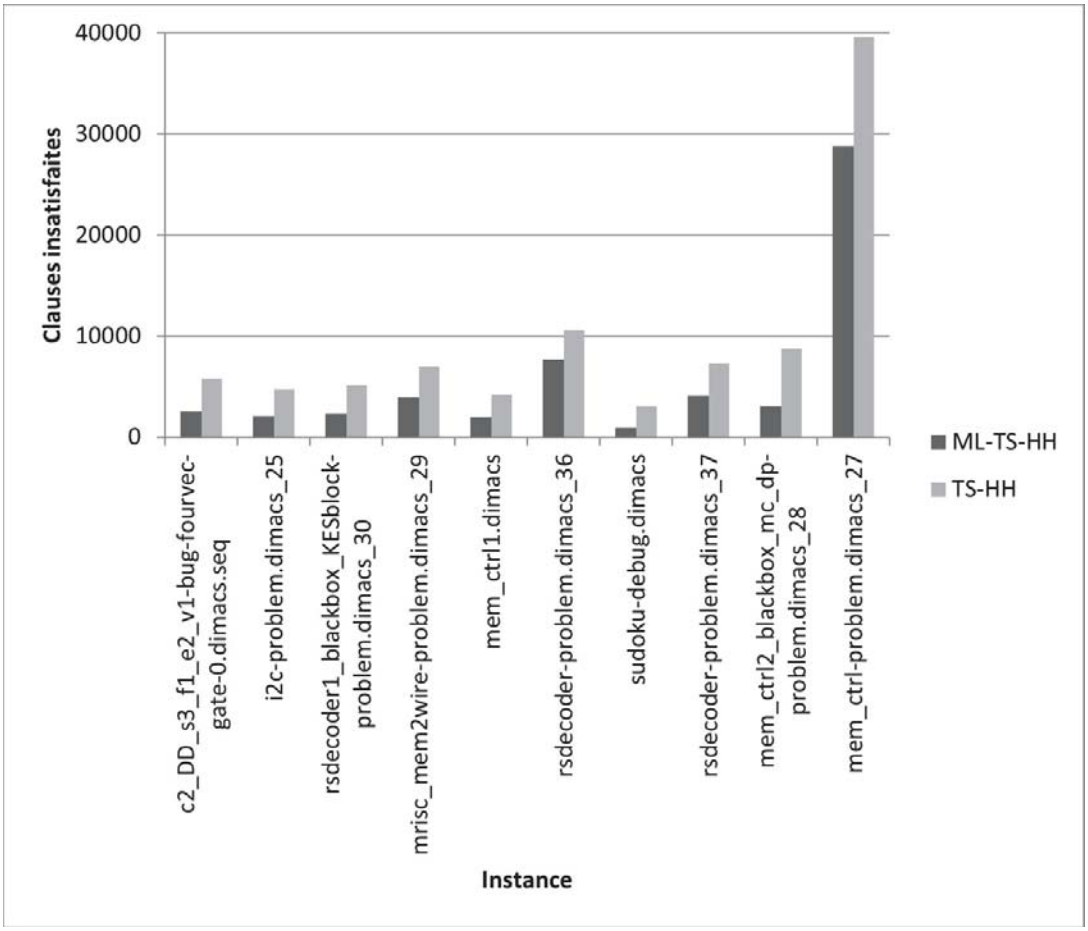



Figure 5.1 – ML-TS-HH Vs TS-HH

TABLE 5.2 – Résultats de la comparaison entre les cinq méthodes sur les instances industriels (a)

| **Benchmark** | **Variables** | **Clauses** | **ML-SyTS -HH** | **ML-TS -HH** | **ML-SCF -HH** | **ML-CF -HH** | **ML-R -HH** | **Best compet.** |
|---|---|---|---|---|---|---|---|---|
| c1_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred | 391897 | 989885 | **6** | 33 | 10 | 75 | 14 | 4 |
| c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred | 400085 | 1121810 | **1375** | 2553 | 4789 | 5627 | 5351 | 4 |
| c4_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred | 797728 | 2011216 | **512** | 597 | 663 | 795 | 1338 | 8 |
| c4_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred | 448465 | 1130672 | **233** | 235 | 462 | 302 | 271 | 4 |
| c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred | 200944 | 540984 | **8** | 8 | 8 | 8 | 8 | 8 |
| c5_DD_s3_f1_e1_v2-bug-gate-0.dimacs.seq.filtred | 200944 | 540984 | **8** | 8 | 8 | 10 | 8 | 8 |
| c6_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred | 298058 | 795900 | **1391** | 1432 | 1406 | 1461 | 1471 | 8 |
| mem_ctrl1.dimacs.filtred | 1128648 | 4422185 | **1526** | 1982 | 1634 | 2662 | 4144 | 1 |
| mem_ctrl2_blackbox_mc_dp-problem.dimacs_28.filtred | 1974822 | 6795573 | **2148** | 3052 | 4753 | 4991 | 8518 | 3 |
| mem_ctrl-problem.dimacs_27.filtred | 4426323 | 15983633 | **10368** | 28805 | 22291 | 23565 | 533343 | 393801 |
| mrisc_mem2wire-problem.dimacs_29.filtred | 844900 | 2905976 | **1265** | 3951 | 1462 | 1920 | 3323 | 1 |
| divider-problem.dimacs_11.filtred | 215964 | 709377 | **1799** | 1991 | 2049 | 3689 | 2841 | 9 |
| divider-problem.dimacs_2.filtred | 228874 | 750705 | **1800** | 1807 | 2043 | 2106 | 2211 | 2 |
| divider-problem.dimacs_5.filtred | 228874 | 750705 | **1163** | 2265 | 2756 | 3866 | 5021 | 11 |
| divider-problem.dimacs_8.filtred | 246943 | 810105 | **1852** | 1967 | 1939 | 2814 | 2937 | 10 |
| dividers10.dimacs.filtred | 45552 | 162874 | **252** | 285 | 317 | 386 | 395 | 2 |
| dividers_multivec1.dimacs.filtred | 106128 | 397650 | **1402** | 1562 | 1589 | 1614 | 1772 | 2 |
| i2c-problem.dimacs_25.filtred | 521672 | 1581471 | **1018** | 2091 | 1726 | 1764 | 1743 | 2 |
| i2c-problem.dimacs_26.filtred | 397668 | 1205454 | **1123** | 1167 | 1185 | 1360 | 1472 | 2 |
| SM_AS_TOP_buggy1.dimacs.filtred | 145900 | 694438 | **905** | 1262 | 2954 | 1065 | 1129 | 84 |
| SM_MAIN_MEM_buggy1.dimacs.filtred | 870975 | 3812147 | **6922** | 8196 | 8988 | 7653 | 8497 | 577 |
| SM_RX_TOP.dimacs.filtred | 235456 | 934091 | **766** | 822 | 913 | 961 | 943 | 6 |
| fpu_multivec1-problem.dimacs_14.filtred | 257168 | 928310 | **1948** | 2541 | 2334 | 4516 | 3619 | 2 |
| rsdecoder-debug.dimacs | 847501 | 2223029 | **3015** | 4135 | 10637 | 10570 | 12282 | 1 |
| sudoku-debug.dimacs | 1304121 | 1554820 | **756** | 967 | 1010 | 2547 | 809 | 1 |
| wb-debug.dimacs | 399591 | 621323 | **166** | 498 | 300 | 226 | 378 | 28 |

TABLE 5.3 – Résultats de la comparaison entre les cinq méthodes sur les instances industriels (b)

| **Benchmark** | **Variables** | **Clauses** | **ML-SyTS -HH** | **ML-TS -HH** | **ML-SCF -HH** | **ML-CF -HH** | **ML-R -HH** | **Best compet.** |
|---|---|---|---|---|---|---|---|---|
| rsdecoder1_blackbox_CSEEblock-problem.dimacs_32.filtred | 277950 | 806460 | **188** | 2728 | 1242 | 2295 | 2358 | 13 |
| rsdecoder1_blackbox_KESblock-problem.dimacs_30.filtred | 707330 | 1106376 | **1347** | 2323 | 2296 | 2389 | 2428 | 659 |
| rsdecoder2.dimacs.filtred | 415480 | 1632526 | **150** | 475 | 390 | 302 | 320 | 1 |
| rsdecoder4.dimacs.filtred | 237783 | 933978 | **51** | 65 | 147 | 152 | 172 | 4 |
| rsdecoder5.dimacs.filtred | 238290 | 936006 | **47** | 55 | 113 | 183 | 174 | 8 |
| rsdecoder6.dimacs.filtred | 238290 | 936006 | **55** | 219 | 187 | 199 | 111 | 80 |
| rsdecoder_fsm2.dimacs.filtred | 238290 | 936006 | **50** | 56 | 122 | 110 | 98 | 2 |
| rsdecoder_multivec1.dimacs.filtred | 394446 | 1626312 | **105** | 1438 | 877 | 932 | 826 | 4 |
| rsdecoder_multivec1-problem.dimacs_33.filtred | 627993 | 2125620 | **305** | 2757 | 361 | 485 | 664 | 1011 |
| rsdecoder-problem.dimacs_31.filtred | 1197376 | 3863287 | **705** | 804 | 1196 | 1161 | 3148 | 3 |
| rsdecoder-problem.dimacs_36.filtred | 1220616 | 3938467 | **711** | 7737 | 1170 | 1219 | 3884 | 1 |
| rsdecoder-problem.dimacs_37.filtred | 1513544 | 4909231 | **1014** | 4060 | 1239 | 4142 | 1543 | 71 |
| rsdecoder-problem.dimacs_38.filtred | 1198012 | 3865513 | **764** | 5090 | 1162 | 1137 | 1063 | 24 |
| rsdecoder-problem.dimacs_39.filtred | 1199602 | 3868693 | **825** | 1068 | 1262 | 1286 | 1350 | 1 |
| rsdecoder-problem.dimacs_40.filtred | 1220616 | 3938467 | **827** | 1259 | 1141 | 1240 | 1132 | 1 |
| rsdecoder-problem.dimacs_41.filtred | 1186710 | 3829036 | **755** | 24310 | 1155 | 1190 | 1170 | 153 |
| wb1.dimacs.filtred | 49525 | 140091 | **342** | 379 | 382 | 363 | 370 | 218 |
| wb2.dimacs.filtred | 49490 | 140056 | **725** | 780 | 773 | 773 | 758 | 588 |
| wb_4m8s1.dimacs.filtred | 463080 | 1759150 | **1465** | 2040 | 1512 | 1591 | 1579 | 460 |
| wb_4m8s3.dimacs.filtred | 463080 | 1759150 | **1514** | 1990 | 1671 | 1658 | 1632 | 8 |
| wb_4m8s4.dimacs.filtred | 463080 | 1759150 | **1431** | 1591 | 1527 | 1592 | 1537 | 14667 |
| wb_4m8s-problem.dimacs_47.filtred | 2691648 | 8517027 | **12720** | 18180 | 16466 | 17736 | 17723 | 59928 |
| wb_4m8s-problem.dimacs_48.filtred | 2766036 | 8774655 | **15297** | 26489 | 26297 | 19836 | 19615 | 62129 |
| wb_4m8s-problem.dimacs_49.filtred | 2785108 | 8812799 | **15676** | 29677 | 17444 | 20081 | 25707 | 63548 |
| wb-problem.dimacs_45.filtred | 309491 | 806440 | 149 | 289 | 157 | 155 | **145** | 14 |
| wb-problem.dimacs_46.filtred | 300846 | 789283 | **638** | 769 | 691 | 677 | 679 | 476 |

TABLE 5.4 – Résultats statistiques de la comparaison entre les cinq méthodes sur les benchmarks industriel de la compétition Max-SAT (a-b)

| Methods | Min | 1st. Qu | Median | Mean | 3rd Qu | Max |
|---|---|---|---|---|---|---|
| **ML-SyTS-HH** | **99.4823%** | **99.8506%** | **99.9353%** | **99.8987%** | **99.9802%** | **99.9993%** |
| **ML-TS-HH** | 99.3651% | 99.785% | 99.8693% | 99.8378% | 99.9641% | 99.9985% |
| **ML-SCF-HH** | 99.448% | 99.7924% | 99.9128% | 99.8543% | 99.9699% | 99.9989% |
| **ML-CF-HH** | 99.448% | 99.7739% | 99.9076% | 99.8369% | 99.9690% | 99.9985% |
| **ML-R-HH** | 96.6631% | 99.7574% | 99.9002% | 99.7218% | 99.9687% | 99.9985% |

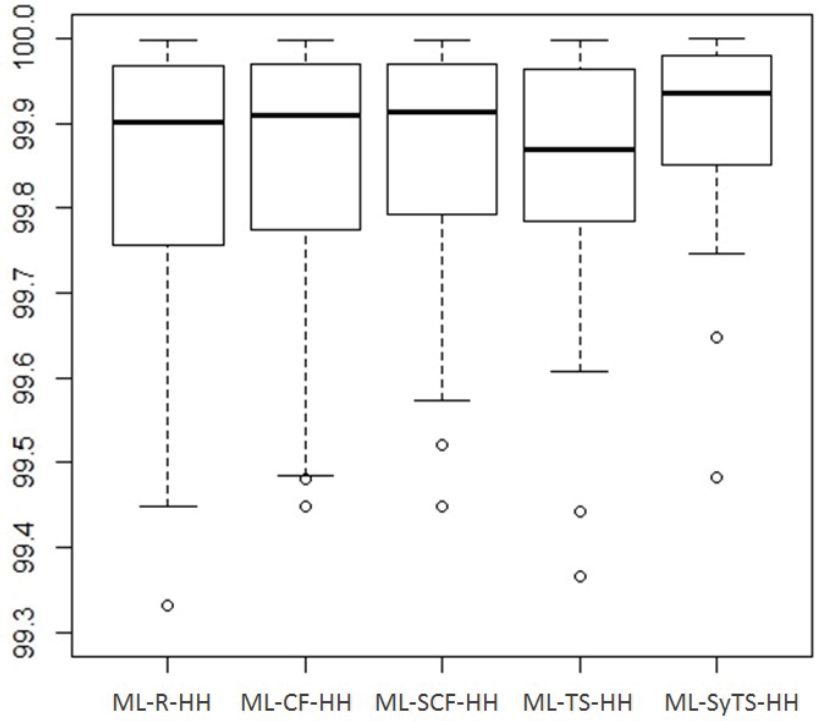



FIGURE 5.2 – Box-plot de la comparaison des cinq méthodes sur les instances de la compétition Max-SAT

Comme il est montré dans le box-plot de la figure 5.2, nous pouvons voir que ML-CF-HH est presque similaire a ML-R-HH, cela indique que le mécanisme d'apprentissage additif de choice function stagne surtout quand il se rapproche de l'optimum global, ce qui rend la probabilité d'améliorer une solution candidate très basse. Par contre nous pouvons remarquer une bonne amélioration concernant ML-SCF-HH. Cela peut être expliqué par le fait que l'aléatoire aide à perturber les valeurs additives des poids de choice function d'une manière qui améliore sa performance. Cette conclusion est confirmée en observant les résultats de ML-TS-HH.

ML-TS-HH a obtenu de meilleurs résultats que ML-CF-HH. Cela montre qu'une stratégie de sélection probabiliste a une meilleure performance qu'un mécanisme d'apprentissage additif. Cependant, ML-SCF-HH est meilleure que ML-TS-HH. Nous pouvons dire dans ce cas, que le mécanisme stochastique améliore vraiment le mécanisme d'apprentissage additif, qui prend en charge la synergie entre les heuristiques de bas niveau. La méthode de sélection Thomp-

Table 5.5 – Résultats de l'analyse ANOVA

| Hyper-heuristic methods | df | SS | MS | F-value | P-value |
|---|---|---|---|---|---|
| ML-R-HH Vs ML-CF-HH | 1 | 9.61e+10 | 9.61e+10 | 27.07 | **3.52e-6** |
| ML-R-HH Vs ML-SCF-HH | 1 | 8.157e+10 | 8.157e+10 | 21.27 | **2.71e-5** |
| ML-R-HH Vs ML-TS-HH | 1 | 7.566e+10 | 7.566e+10 | 19.15 | **5.99e-5** |
| ML-R-HH Vs ML-SyTS-HH | 1 | 4.225e+10 | 4.225e+10 | 9.174 | **3.85e-3** |
| ML-CF-HH Vs ML-SCF-HH | 1 | 1.406e+9 | 1.406e+9 | 975.06 | **< 2.0e-16** |
| ML-CF-HH Vs ML-TS-HH | 1 | 1.074e+9 | 1.074e+9 | 134.8 | **6.21e-16** |
| ML-CF-HH Vs ML-SyTS-HH | 1 | 1.316e+9 | 1.316e+9 | 409.0 | **<2.0e-16** |
| ML-SCF-HH Vs ML-TS-HH | 1 | 1.137e+9 | 1.137e+9 | 127.1 | **1.84e-15** |
| ML-SCF-HH Vs ML-SyTS-HH | 1 | 1.428e+9 | 1.428e+9 | 442.7 | **<2.0e-16** |
| ML-TS-HH Vs ML-SyTS-HH | 1 | 2.028e+9 | 2.028e+9 | 138.3 | **3.85e-16** |

son Sampling se base sur les performances individuelles des heuristiques de bas niveau.

Finalement, les résultats expérimentaux indiquent que ML-SyTS-HH est la plus robuste entre les 5, nous pensons que cela est dû principalement à sa stratégie de sélection probabiliste qui prend en charge la synergie entre les heuristiques de bas niveau, cela confirme aussi que la coopération permet de combler les faiblesses de certaines heuristiques de bas niveau par la force d'autres. Les résultats soulignés de la ML-SyTS-HH sont ceux qui égalent ou dépassent les meilleurs résultats des méthodes incomplètes de la compétition Max-SAT 2014. Cela est dû au paradigme multiniveau et le principe de la synergie qui ont permis une bonne exploration de l'espace de recherche. Donc on peut dire que la coopération entre des méthodes de l'état de l'art de Max-SAT peut rivaliser avec de nouvelles méthodes sophistiquées et plus élaborées spécialement conçues pour le problème Max-SAT.

### 5.6.3 L'analyse statistique ANOVA

Afin de mieux interpréter les résultats statistiques, nous avons utilisé l'analyse ANOVA. La table 5.5 montre les résultats des 10 tests ANOVA où la colonne **df** représente le degré de liberté, la colonne **SS** représente la somme des carrés, la colonne **MS** représente le carré de la moyenne, la **F-value** représente le F-statistique, et la colonne **P-value** en gras représente l'interprétation du résultat de l'analyse.

Les valeurs de P-value sont tous inférieur à 0.05, cela indique que les valeurs produites par les 5 méthodes sont très différentes l'une de l'autre. Cela signifie que notre hyper-heuristique ML-SyTS-HH est statistiquement meilleure que les autres méthodes, ce qui confirme les conclusions avancées précédemment.

## 5.7 Conclusion

Dans ce chapitre, nous exposons notre approche hyper-heuristique qui combine le paradigme multiniveau avec une hyper-heuristique Thompson Sampling modifiée qui prend en compte la synergie entre les heuristiques de bas niveau. Nous avons également utilisé, comme heuristiques de bas niveau, des méthodes bien connues de l'état de l'art de Max-SAT, à savoir GSAT, WalkSAT, HSAT, SLS, VNS et Novelty. Nous avons comparé notre approche avec quatre autres hyper-heuristiques dotée du paradigme multiniveau : l'hyper-heuristique de sélection aléatoire, l'hyper-heuristique choice function, l'hyper-heuristique stochastique choice function et l'hyper-heuristique Thompson Sampling originale. Les tests ont été faits sur des benchmarks industriels de très grandes tailles, de la compétition Max-SAT 2014. Les résultats montrent la supériorité de notre approche, ce qui prouve qu'elle exploite plus efficacement les heuristiques de bas niveau. Cela donc confirme que les faiblesses de certaines heuristiques de bas niveau sont compensées par la force d'autres heuristiques. La comparaison des résultats de ML-SyTS-HH avec les meilleurs résultats obtenus, par les méthodes incomplètes, de la compétition Max-SAT 2014 montre que la coopération entre des méthodes de l'état de l'art de Max-SAT peut rivaliser avec de nouvelles méthodes sophistiquées spécialement conçues pour le problème Max-SAT.

# Chapitre 6

# L'hyper-heuristique Synergie Thompson Sampling pour le problème de la sélection d'attributs

## 6.1 Introduction

Afin de classifier des données multidimensionnelles, la sélection d'attributs est une étape cruciale qui élimine les attributs non pertinents, et donc qui améliore l'efficacité de la classification. Comme la sélection d'attributs est un problème NP-difficile [Blum and Rivest, 1989], plusieurs heuristiques et métaheuristiques ont été appliquées. Dans ce chapitre, nous vous proposons notre méthode appelée **Hyper-heuristique Synergie Thompson Sampling** pour le problème de la sélection d'attributs. Dans ce qui suit, nous allons détailler les composantes de notre méthode et exposer les résultats obtenus.

## 6.2 Modélisation

Dans ce qui suit, nous allons détailler l'encodage de la solution, la fonction objectif, l'algorithme de classification utilisé et les heuristiques de bas niveau.

### 6.2.1 Représentation de la solution

Une solution, comme dans la figure 6.1, est représentée en utilisant un vecteur binaire $X$ de taille $N, N$ étant le nombre d'attributs dans l'instance. Chaque élément $X_i$ représente l'attribut $i$, et peut recevoir la valeur 0 ou 1. Si $X_i = 0$ cela veut dire que l'attribut $i$ ne sera pas pris en considération durant le processus de classification. Par contre, si $X_i = 1$, alors l'attribut $i$ est pris en compte pour la classification.

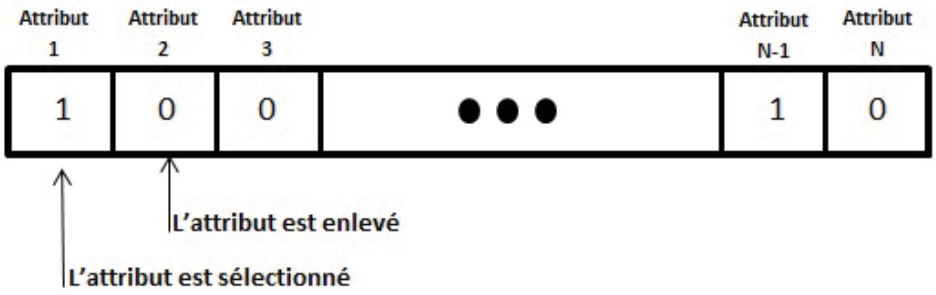



FIGURE 6.1 – Représentation d'une solution

### 6.2.2 La fonction objectif

La qualité de la solution (*fitness*) est mesurée par la fonction objectif. Dans le problème de sélection d'attributs, cela consiste à maximiser la précision (*accuracy*) du taux de classification avec un ensemble réduit d'attributs. Dans notre cas, nous avons utilisé la validation croisée avec 10 échantillons (*10-fold cross validation*) qui est la technique classique pour mesurer la précision d'une méthode d'apprentissage. Dans ce cas, les données de l'instance sont divisées aléatoirement en dix sous-ensembles de même taille. Durant chaque exécution, un des sous-ensembles est retenu comme ensemble de test et le reste sera utilisé pour l'apprentissage. Le processus est itéré dix fois, ou chaque sous-ensemble est utilisé une seule foi pour le test et neuf fois pour l'apprentissage. La précision de l'algorithme de classification est évaluée en calculant le ratio du nombre d'instances correctement classifiées par le nombre total d'instances suivant la formule suivante :

$$Precision = \frac{tp + tn}{tp + fn + fp + tn}$$

Où, **tp** est le vrai positif (*true positive*), **tn** est le vrai négatif (*true negative*), **fp** est le faux positif (*false positive*) et **fn** est le faux négatif (*false negative*). Ces valeurs sont récupérées à partir d'un classifieur.

### 6.2.3 Le classifieur

Dans notre travail, nous avons utilisé l'algorithme 1-NN afin d'évaluer la qualité du sous ensemble d'attributs retenu.

Ce classifieur est basé sur la notion de proximité (voisinage). La classe assignée au point $x$ sera celle de son plus proche voisin. Ce classifieur est connu pour être stable et ne pose aucune hypothèse sur la forme des classes à apprendre, de plus cette méthode est simple et son pouvoir prédictif est souvent bon [Montazeri, 2016]. Ce qui le rend le plus adapté pour pouvoir comparer les performances des hyper-heuristique.

### 6.2.4 Les heuristiques de bas niveau

Dans notre méthode nous avons utilisé six heuristiques perturbatrices :

- L'heuristique $h_1$ : cette heuristique génère à chaque itération $N$ voisins. Cela est accomplie en flippant chaque attribut exactement une fois dans le vecteur de solution. Flipper une variable veut dire l'ajouter ou l'enlever du sous ensemble d'attributs retenu. Le meilleur voisin trouvé est comparé à la meilleure solution trouvée. S'il y a amélioration, alors la solution est retenue. Ce processus est réitéré **nbIter** fois.

- L'heuristique $h_2$ : similaire à $h_1$ qui génère les $N$ voisins, $h_2$ s'arrête dès qu'elle trouve un voisin qui améliore la solution. Ce processus est réitéré **nbIter** fois.

- L'heuristique $h_3$ : Cette heuristique sélectionne à chaque itération un attribut aléatoirement et le flippe. S'il y a amélioration, alors la nouvelle solution est retenue. Ce processus est réitéré **nbIter** fois.

- L'heuristique $h_4$ : cette heuristique utilise un opérateur de croisement entre la meilleure solution trouvée et la solution courante.

- L'heuristique $h_5$ : cette heuristique utilise un opérateur de croisement entre la meilleure solution trouvée et une solution générée aléatoirement.

- L'heuristique $h_6$ : c'est une recherche locale stochastique (SLS). C'est une métaheuristique qui est bien connue qui a été utilisée avec succès sur diffèrent problèmes d'optimisation.

### 6.2.5 L'hyper-heuristique Synergie Thompson Sampling

Nous avons utilisé dans cette partie, l'hyper-heuristique Synergie Thompson Sampling (SyTS-HH) qui est détaillée dans le chapitre précèdent, et qui a prouvé son efficacité pour le problème Max-SAT.

## 6.3 Les résultats expérimentaux

Dans cette section, nous allons présenter l'étude expérimentale mis en œuvre pour tester les performances de notre approche. Dans ce qui suit, nous allons donner un bref descriptif des benchmarks utilisés, de la phase de paramétrage et de l'environnement de développement. Ensuite, nous exposons le comparatif des résultats de l'hyper-heuristique Thompson Sampling (TS-HH) et de l'hyper-heuristique Synergie Thompson Sampling (SyTS-HH), afin de voir l'impact de l'aspect synergie. Enfin nous comparons nos résultats avec ceux de [Montazeri, 2016], étant le travail le plus récent sur les hyper heuristique pour le problème de sélection d'attributs.

### 6.3.1 Description des benchmarks

Nous avons utilisé 29 ensembles de données à partir de l'UCI Learning Repository[1]. Les ensembles de données sont décrites dans la table 6.1, où $N$ représente

[1]https ://archive.ics.uci.edu/ml/index.php

TABLE 6.1 – Description benchmarks UCI

| **Dataset** | ***N*** | **Nombre de classe** | **Nombre d'instances** |
|---|---|---|---|
| Audiology | 70 | 24 | 226 |
| Arrhythmia | 263 | 13 | 452 |
| Breast-cancer | 10 | 2 | 286 |
| Breast-cancer-Wisconsin | 10 | 2 | 699 |
| Dermatology | 35 | 6 | 366 |
| Diabetes | 9 | 2 | 768 |
| Ecohardiogram | 9 | 2 | 132 |
| Glass | 10 | 7 | 214 |
| Habermans | 4 | 2 | 306 |
| Heart-c | 14 | 5 | 303 |
| Heart-stat | 14 | 2 | 270 |
| Heart-swiss | 13 | 5 | 123 |
| Hepatitis | 20 | 2 | 155 |
| Horse-colis | 28 | 2 | 300 |
| Ionosphere | 35 | 2 | 351 |
| Iris | 5 | 3 | 150 |
| Kr-vs-kp | 37 | 2 | 3196 |
| Liver-disorders | 7 | 2 | 345 |
| Lung-cancer | 57 | 3 | 32 |
| Lymph | 19 | 4 | 148 |
| Mammographic-mass | 6 | 2 | 961 |
| Molecular-splice | 61 | 3 | 3190 |
| New-thyroid | 6 | 3 | 215 |
| Sonar | 61 | 2 | 208 |
| Soybean | 36 | 19 | 683 |
| Spectf | 45 | 2 | 349 |
| Synthetic | 61 | 6 | 600 |
| Vehicle | 19 | 4 | 846 |
| Vowel | 14 | 11 | 990 |

le nombre d'attributs. Pour notre étude nous avons choisis différents domaines tel que : médical, sciences sociales, jeux, physique etc.

### 6.3.2 Environnement expérimental

Notre méthode a été programmée en Java, et nous avons utilisé l'outil Weka 3.9.2. Les tests ont été faits sur une machine avec le processeur intel i7 avec 8G de RAM.

### 6.3.3 Les paramètres

Les différents paramètres ont été fixés empiriquement :

- **w** : représente la taille de la fenêtre glissante de l'hyper-heuristique, elle est fixée à 10.

- **maxiter** : représente le nombre d'itération de l'hyper-heuristique, elle est fixée à 200.
- **nbIter** : représente le nombre d'itérations des heuristiques de bas niveau $h_1, h_2$ et $h_3$, il est fixé à 10.
- **wp** : c'est la probabilité de marche de SLS, elle est fixée à 0.3.

A cause de l'aspect aléatoire des algorithmes, nous avons retenu la moyenne de 10 exécutions pour chaque méthode pour chaque benchmark.

### 6.3.4 Thompson Sampling Vs Synergie Thompson Sampling

Dans cette section, nous exposons la comparaison entre l'hyper-heuristique Thomson Sampling et l'hyper-heuristique Synergie Thompson Sampling. Les résultats sont montrés dans les tableaux 6.2 et 6.3, où le meilleur résultat obtenu pour chaque benchmark est présenté en gras. Les résultats sont exprimés par la **précision de la classification** et le **nombre des attributs retenu**. Pour mieux visualiser les résultats des diagrammes box-plot sont donnés dans la figure 6.2.

Table 6.2 – Résultats de la meilleure précision et du meilleur nombre d'attributs de la comparaison entre TS-HH et SyTS-HH

| | | Meilleure précision | | Meilleur nb d'attributs | |
|---|---|---|---|---|---|
| Dataset | Sans FS | TSHH | SyTSHH | TSHH | SyTSHH |
| Audiology | 0.62 | 0.84 | **0.86** | 19 | **17** |
| Breast-cancer | 0.72 | 0.76 | 0.76 | 4 | 4 |
| Breast-cancer-wisconsin | 0.95 | 0.97 | 0.97 | 5 | 5 |
| Dermathology | 0.96 | 0.98 | 0.98 | 11 | **10** |
| Diabetes | 0.70 | 0.70 | **0.71** | 4 | **3** |
| Ecohardiogram | 0.64 | 0.77 | 0.77 | 2 | 2 |
| Glass | 0.80 | 0.79 | 0.79 | 5 | 5 |
| Habermans | 0.67 | 0.75 | 0.75 | 2 | 2 |
| Heart-c | 0.76 | 0.83 | 0.83 | 3 | 3 |
| Heart-stat | 0.75 | 0.82 | **0.84** | 9 | **3** |
| Heart-swiss | 0.36 | 0.46 | 0.46 | 4 | 4 |
| Hepatitis | 0.80 | 0.89 | **0.91** | **8** | 10 |
| Horse-colis | 0.77 | 0.87 | **0.91** | 9 | **9** |
| Ionosphere | 0.87 | 0.94 | **0.96** | **8** | 10 |
| Iris | 0.95 | 0.96 | 0.96 | 2 | 2 |
| Liver-disorders | 0.63 | 0.68 | **0.68** | 5 | **4** |
| Lung-cancer | 0.44 | 0.90 | **0.93** | 13 | **11** |
| Lymph | 0.81 | 0.88 | **0.88** | 14 | **8** |
| Mammographic-mass | 0.75 | **0.83** | **0.83** | **2** | **2** |
| New-thyroid | 0.97 | 0.97 | 0.97 | 3 | 3 |
| Sonar | 0.87 | 0.96 | **0.98** | 24 | **21** |
| Soybean | 0.87 | 0.95 | **0.95** | 17 | **16** |
| Spectf | 0.85 | 0.93 | **0.95** | 19 | **9** |
| Vehicle | 0.69 | 0.74 | **0.74** | 12 | **8** |
| Vowel | 0.99 | **0.99** | **0.99** | **9** | **9** |

TABLE 6.3 – Résultats de la précision moyenne et de la moyenne du nombre d'attributs de la comparaison entre TS-HH et SyTS-HH

| | | Moyenne précision | | Moyenne nb attributs | |
|---|---|---|---|---|---|
| Dataset | Sans FS | TSHH | SyTSHH | TSHH | SyTSHH |
| Audiology | 0.62 | 0.80 | **0.86** | 23.3 | 18.6 |
| Breast-cancer | 0.72 | 0.75 | **0.76** | 4.3 | 4.8 |
| Breast-cancer-wisconsin | 0.95 | 0.96 | **0.97** | 5.6 | 5.6 |
| Dermathology | 0.96 | 0.97 | **0.98** | 15.7 | 13.4 |
| Diabetes | 0.70 | 0.70 | **0.71** | 4.3 | 3 |
| Ecohardiogram | 0.64 | 0.75 | **0.77** | 2.4 | 2 |
| Glass | 0.80 | 0.78 | **0.79** | 5.4 | 5 |
| Habermans | 0.67 | 0.75 | 0.75 | 2 | 2 |
| Heart-c | 0.76 | 0.81 | **0.83** | 6.5 | 3.6 |
| Heart-stat | 0.75 | 0.80 | **0.84** | 6.7 | 3 |
| Heart-swiss | 0.36 | 0.43 | **0.46** | 5.3 | 4 |
| Hepatitis | 0.80 | 0.86 | **0.91** | 8.6 | 9.9 |
| Horse-colis | 0.77 | 0.86 | **0.91** | 10 | 8 |
| Ionosphere | 0.87 | 0.93 | **0.96** | 12.8 | 9.2 |
| Iris | 0.95 | 0.95 | **0.96** | 2.1 | 2 |
| Liver-disorders | 0.63 | 0.62 | **0.68** | 2.8 | 4 |
| Lung-cancer | 0.44 | 0.82 | **0.92** | 13.9 | 11.8 |
| Lymph | 0.81 | 0.86 | **0.88** | 9.8 | 9 |
| Mammographic-mass | 0.75 | 0.82 | **0.83** | 2.1 | 2 |
| New-thyroid | 0.97 | 0.97 | 0.97 | 3 | 3 |
| Sonar | 0.87 | 0.93 | **0.97** | 24.7 | **22.1** |
| Soybean | 0.87 | 0.91 | **0.95** | 17.4 | 17.6 |
| Spectf | 0.85 | 0.91 | **0.94** | 19.7 | **12.8** |
| Vehicle | 0.69 | 0.72 | **0.74** | 11.4 | **8** |
| Vowel | 0.99 | 0.99 | **0.99** | 9.1 | **8.8** |

Nous pouvons voir clairement que la SyTS-HH donne de meilleurs résultats que la TS-HH dans 23 benchmarks sur 25 testés. Cela prouve que l'intégration de la synergie entre les heuristiques de bas niveau booste la performance de l'hyper-heuristique. Cela confirme, encore une fois que la coopération permet de combler les faiblesses de certaines heuristiques par la force des autres. Cela aussi permet une meilleure exploration de l'espace de recherche, et avoir un bon équilibre entre intensification et diversification. Nous pouvons aussi remarquer que la performance de SyTS-HH est stable sur les 10 exécutions par rapport à la TS-HH.

### 6.3.5 Comparaison entre SyTS-HH et l'hyper-heuristique génétique

Dans cette section, nous présentons la comparaison entre SyTS-HH et l'hyper heuristique génétique de [Montazeri, 2016]. L'hyper-heuristique génétique

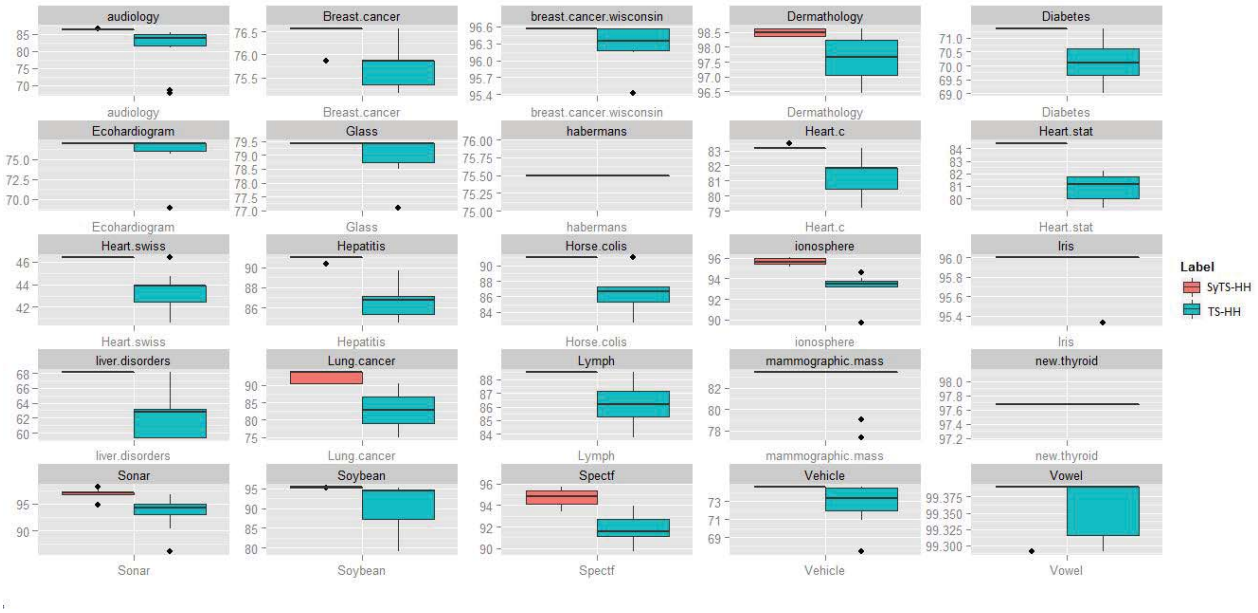



FIGURE 6.2 – Box-plot des résultats de la comparaison entre TS-HH et SyTS-HH

s'est montrée plus efficace en comparaison avec les méthodes les plus récentes de la sélection d'attributs présentes dans l'état de l'art [Montazeri, 2016]. La première, appelée *Hyper-Heuristic Feature Selection* (HHFS), choisit une heuristique de bas niveau selon une stratégie de sélection génétique. Elle démarre avec une population de chromosomes, où chaque chromosome représente une séquence d'heuristiques de bas niveau. Des opérateurs génétiques sont appliqués afin de générer la population suivante. La seconde hyper-heuristique, appelée *Adaptive Hyper-Heuristic Feature Selection* (AHHFS), est une extension de la première, où la taille des chromosomes n'est pas constante, elle change à chaque génération. Dans les deux méthodes, la fonction d'évaluation se base sur la validation croisée avec 10 échantillons (*10-fold cross validation*) avec la classifieur 1-NN. Les résultats sont montrés dans les tables 6.4 et 6.5. Pour une meilleure visualisation les figures 6.3 et 6.4 sont présentées.

On peut voir que la performance de SyTS-HH dépasse celle des deux hyper-heuristiques génétique sur presque tous les benchmarks, et cela sur la précision et le nombre d'attributs retenus. Le benchmark Arrhythmia et kr-vs-kp sont les deux exceptions. Cependant, dans le premier, la différence de la précision est de seulement **2.9%** , par contre, la moyenne du nombre d'attributs retenus dans notre approche est de **63.7** par rapport à **138.5** pour HHFS, ce qui est moins que la moitié. Pour le deuxième benchmark, la moyenne du nombre d'attributs retenus dans notre approche est de **20.1** par rapport à **6.1** pour AHHFS, par contre la précision de notre approche est de **98.5%** par rapport à **63%** pour AHHFS.

Nous pouvons dire que l'hyper-heuristique génétique cherche la meilleure séquence des heuristiques de bas niveau en les ordonnant au hasard. Par contre, le mécanisme d'apprentissage en ligne de SyTS-HH, permet de construire des séquences d'heuristiques de bas niveau bien plus efficaces, en calculant les scores

TABLE 6.4 – Résultats de la meilleure précision et du meilleur nombre d'attributs de la comparaison entre SyTS-HH, HHFS et AHHFS

| | | Meilleure Précision | | | Meilleur nb attributs | | |
|---|---|---|---|---|---|---|---|
| Dataset | Without FS | SyTSHH | HHFS | AHHFS | SyTSHH | HHFS | AHHFS |
| Audiology | 0.62 | **0.86** | 0.82 | 0.85 | 17 | 33 | 9 |
| Arrhythmia | 0.69 | 0.74 | **0.76** | 0.55 | **63** | 148 | 137 |
| Dermathology | 0.96 | 0.98 | 0.98 | 0.98 | **10** | 29 | 13 |
| Glass | 0.80 | **0.79** | 0.74 | 0.73 | 5 | 5 | 5 |
| Ionosphere | 0.87 | **0.96** | 0.94 | 0.95 | **8** | 12 | 9 |
| Kr-vs-kp | 0.85 | **0.98** | 0.97 | 0.71 | 11 | 20 | 8 |
| Lymph | 0.81 | **0.88** | 0.86 | 0.87 | 8 | 7 | 11 |
| Molecular-splice | 0.75 | **0.90** | 0.90 | 0.90 | **5** | 8 | 8 |
| Sonar | 0.87 | **0.98** | 0.93 | 0.94 | **21** | 26 | 30 |
| Soybean | 0.87 | **0.95** | 0.93 | 0.95 | **16** | 19 | 19 |
| Spectf | 0.85 | **0.95** | 0.89 | 0.90 | **9** | 19 | 17 |
| Synthetic | 0.98 | **1** | 1 | 1 | **22** | 36 | 31 |

TABLE 6.5 – Résultats de la précision moyenne et de la moyenne du nombre d'attributs de la comparaison entre SyTS-HH, HHFS et AHHFS

| | | Moyenne Précision | | | Moyenne nb attributs | | |
|---|---|---|---|---|---|---|---|
| Dataset | Without FS | SyTSHH | HHFS | AHHFS | SyTSHH | HHFS | AHHFS |
| Audiology | 0.62 | **0.86** | 0.80 | 0.82 | **18.6** | 33.5 | 32 |
| Arrhythmia | 0.69 | 0.72 | **0.75** | 0.53 | **63.7** | 138.5 | 134.6 |
| Dermathology | 0.96 | 0.98 | 0.98 | 0.98 | **13.4** | 27 | 22.6 |
| Glass | 0.80 | **0.79** | 0.73 | 0.72 | **5** | 5.8 | 6.3 |
| Ionosphere | 0.87 | **0.96** | 0.93 | 0.93 | **9.2** | 14.5 | 14.5 |
| Kr-vs-kp | 0.85 | **0.98** | 0.96 | 0.63 | 20.1 | 22.7 | 6.1 |
| Lymph | 0.81 | **0.88** | 0.85 | 0.86 | **9** | 11.7 | 12.2 |
| Molecular-splice | 0.75 | **0.90** | 0.90 | 0.90 | **5.1** | 7.6 | 7.5 |
| Sonar | 0.87 | **0.97** | 0.90 | 0.92 | **22.1** | 31.9 | 33.5 |
| Soybean | 0.87 | **0.95** | 0.93 | 0.94 | **17.6** | 20.8 | 21.9 |
| Spectf | 0.85 | **0.94** | 0.88 | 0.89 | **12.8** | 22.8 | 23.7 |
| Synthetic | 0.98 | 0.99 | **1** | **1** | **25.2** | 41.1 | 37.3 |

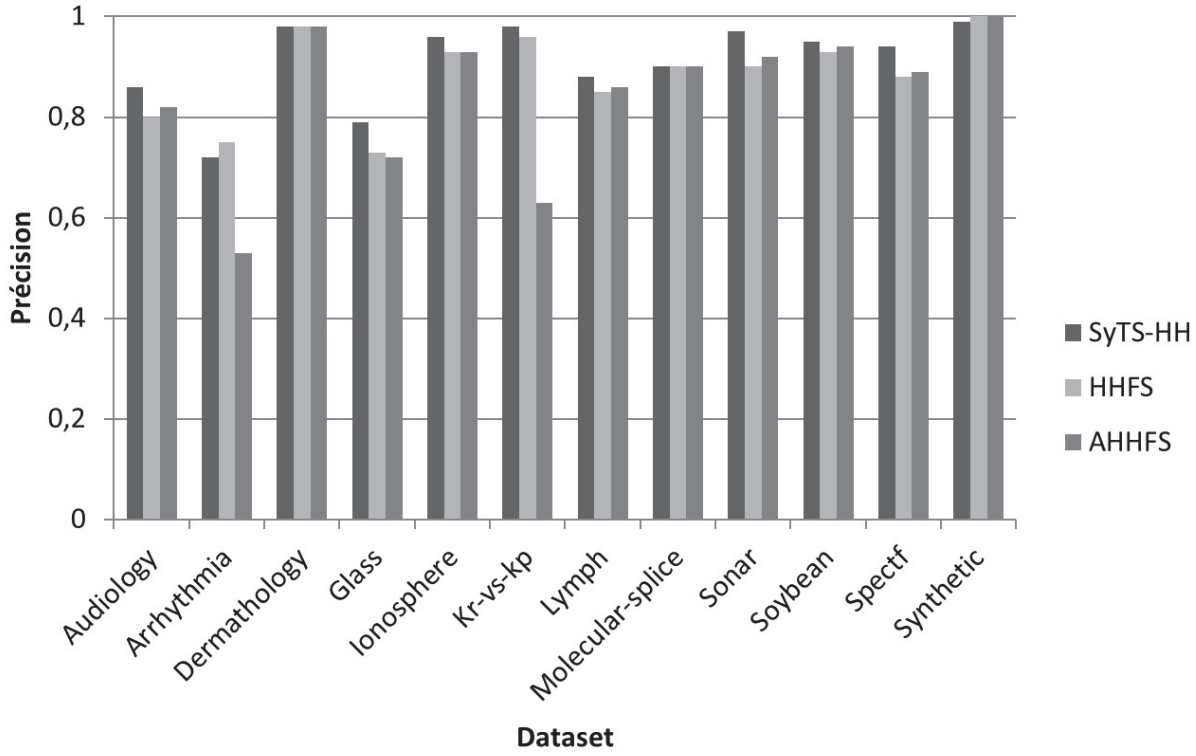



FIGURE 6.3 – Précision moyenne de la comparaison entre SyTS-HH, HHFS et AHHFS

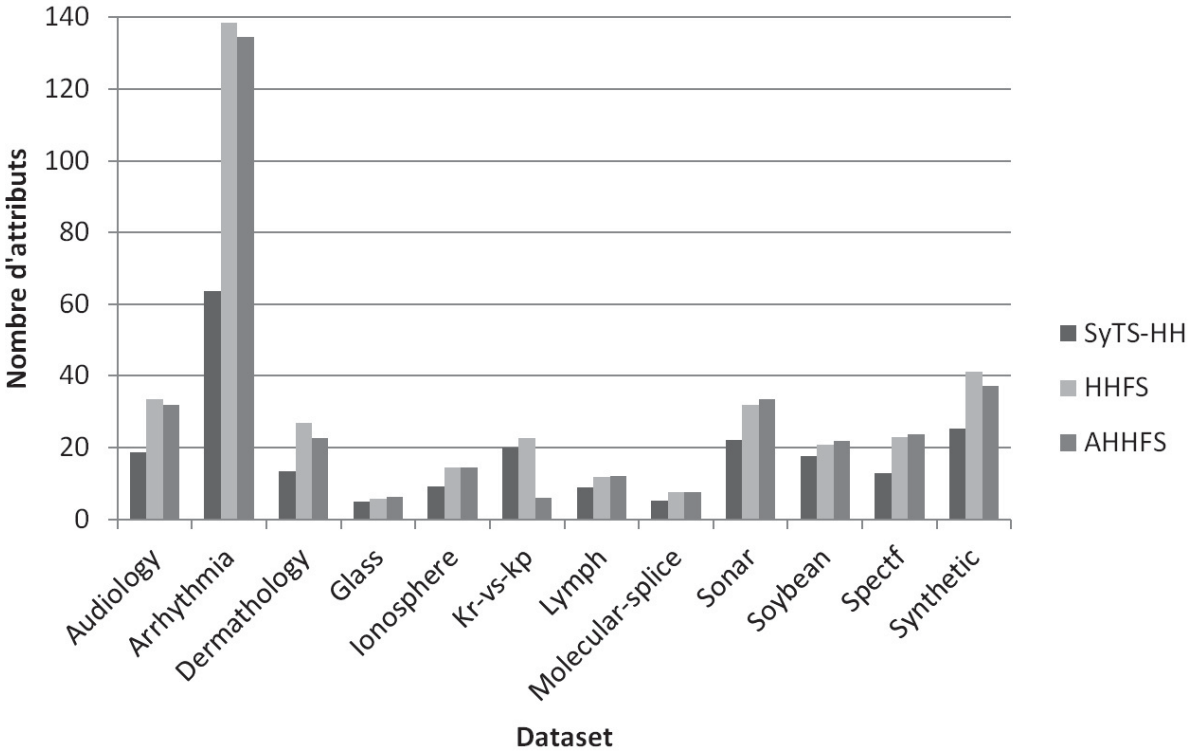



FIGURE 6.4 – Nombre d'attributs moyen de la comparaison entre SyTS-HH, HHFS et AHHFS

de synergie avec les informations récoltées durant l'exécution. Cela permet donc une meilleure exploration de l'espace de recherche, et une meilleure exploitation des heuristiques de bas niveau.

## 6.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode probabiliste pour le problème de sélection d'attributs. Puisque les données brutes peuvent être bruitées, cela peut nuire aux performances des classifieurs, ce qui montre l'importance de la sélection des attributs. Identifier les attributs appropriés améliore considérablement le taux de précision. Puisque la sélection d'attributs est un problème NP-difficile, les hyper-heuristiques sont des méthodes très appropriées. Notre méthode utilise un mécanisme d'apprentissage probabiliste appelé Thompson Sampling, et prend en considération la synergie entre les heuristiques de bas niveau. Les résultats expérimentaux ont montré que SyTS-HH est bien meilleure que TS-HH. Cela prouve l'importance de la synergie, qui permet de trouver de meilleures combinaisons des heuristiques de bas niveau. Nous avons aussi comparé notre méthode avec deux hyper-heuristiques génétiques de [Montazeri, 2016]. L'étude expérimentale montre la supériorité de notre méthode.

# Conclusion Générale

Dans cette thèse, nous nous somme intéressé aux méthodes de résolution hyper-heuristiques et à leur application à deux problèmes NP-difficiles, à savoir le problème Max-SAT et le problème de sélection d'attributs dans la classification.

Pour le problème Max-SAT, à cause de ses instances industrielles de très grandes tailles, nous avons fait appel au paradigme multiniveau. Le paradigme multiniveau contracte récursivement l'instance afin de créer une hiérarchie d'approximations du problème original. C'est approximations sont des échantillons de plus petite taille et donc plus facile à résoudre. Cela est fait en combinant les variables du problème en clusters. Au niveau le plus contracté, une solution initiale est calculée, puis elle sera itérativement raffinée et projetée à chaque niveau en partant du niveau le plus contracté jusqu'au problème original. Donc la meilleure solution trouvée à un niveau est utilisé comme solution initiale pour le niveau suivant.

Dans un premier temps, nous avons développé une approche hybride de sélection d'heuristiques de bas niveau, appelée ***Stochastique Choice-Function***. Elle combine une méthode de sélection utilisant un mécanisme d'apprentissage par renforcement additif, appelée ***choice function***, avec un mécanisme de sélection aléatoire.

Pour l'étude expérimentale, nous avons implémenté quatre hyper heuristiques : l'hyper-heuristique aléatoire (R-HH), l'hyper-heuristique choice function (CF-HH), l'hyper-heuristique stochastique choice function (SCF-HH) et la version multiniveau de la SCF-HH (ML-SCF-HH). Les tests ont été faits sur des instances de petites, moyennes et grandes tailles du problème Max-SAT.

Les résultats montrent que l'hybridation entre choice function et l'aléatoire permet de palier au problème de stagnation constaté dans l'hyper-heuristique choice function. Ils confirment aussi l'avantage apporté par l'intégration du paradigme multiniveau dans l'hyper-heuristique stochastique choice function. La comparaison entre l'hyper-heuristique stochastique choice function multiniveau et les deux méthodes de l'état de l'art GSAT et WalkSAT, montre le net avantage de notre approche notamment sur les instances de très grande taille.

Le travail de [Alanazi and Lehre, 2016] montre les limites des mécanismes d'apprentissage par renforcement additives comme la choice function, notamment quand la probabilité de succès est inférieure à $\frac{1}{2}$. L'hyper-heuristique Thompson Sampling a été proposée comme alternative. Elle évalue les performances individuelles des heuristiques de bas niveau, et essaye de cerner leurs comportements en utilisant la loi de probabilité Beta. Cette loi possède deux paramètres de forme alpha et beta qui représentent respectivement les nombre de succès et d'échecs. Les valeurs de ses paramètres, qui sont gardées et uti-

lisées pour les prédictions, appartiennent à l'intervalle d'une fenêtre glissante afin de ne garder que les valeurs pertinente à chaque étape de la recherche. Cependant cette méthode ne prend pas en considération la synergie entre les heuristiques de bas niveau.

Nous avons proposé une nouvelle hyper-heuristique qui combine le paradigme multiniveau et une amélioration de la stratégie de sélection Thompson Sampling. Elle prend en considération la synergie entre les heuristiques de bas niveau. Cette méthode est appelée l'hyper-heuristique Synergie Thompson Sampling. Les heuristiques de bas niveau utilisées sont des méthodes très connu dans l'état de l'art comme GSAT, WalkSAT, HSAT, VNS, SLS et Novelty. Les résultats montrent que notre approche dépasse les autres hyper-heuristiques sur les benchmarks industriels de très grande taille du problème Max-SAT. Ils confirment que la coopération entres heuristiques de bas niveau permet de compenser les faiblesses de certaines heuristiques par la forces des autres.

La phase de la sélection d'attributs est devenue une étape cruciale dans la classification, notamment avec l'augmentation constante du volume et de la complexité des données. Identifier les attributs les plus pertinents améliore nettement la précision de la classification. Comme ce problème est NP-difficile les hyper-heuristiques sont des méthodes de résolution très appropriées.

Nous avons testé notre nouvelle approche, qui est l'hyper-heuristique Synergie Thompson Sampling, sur les benchmarks UCI de ce problème. Les résultats expérimentaux montrent que la SyTS-HH est meilleure que la TS-HH, ce qui confirme encore une fois que l'aspect synergie est très important pour trouver de meilleures combinaisons entre heuristiques de bas niveau. Nous avons aussi comparé note méthode à l'hyper-heuristique génétique de [Montazeri, 2016]. Les résultats montrent la supériorité de notre approche.

Nous envisageons pour nos travaux futurs :

- D'essayer d'hybrider d'autres méthodes de sélection d'heuristiques de bas niveau et de les tester sur d'autres problèmes NP-difficiles.
- D'explorer d'autres méthodes de contractions, qui combineront les variables d'une manière plus intelligente.
- De trouver d'autres mécanismes d'apprentissage probabiliste qui prendrons en considération d'autres informations, et qui cerneront mieux le comportement des heuristiques de bas niveau à chaque étape de la recherche.

# Bibliographie

[Aghdam et al., 2009] Aghdam, M. H., Ghasem-Aghaee, N., and Basiri, M. E. (2009). Text feature selection using ant colony optimization. *Expert systems with applications*, 36(3) :6843–6853.

[Agrawal and Goyal, 2013] Agrawal, S. and Goyal, N. (2013). Further optimal regret bounds for thompson sampling. In *Artificial intelligence and statistics*, pages 99–107.

[Al-Ani, 2005] Al-Ani, A. (2005). Feature subset selection using ant colony optimization. *International journal of computational intelligence*.

[Alanazi, 2016] Alanazi, F. (2016). Adaptive thompson sampling for hyper-heuristics. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

[Alanazi and Lehre, 2016] Alanazi, F. and Lehre, P. K. (2016). Limits to learning in reinforcement learning hyper-heuristics. In *Evolutionary Computation in Combinatorial Optimization*, pages 170–185. Springer.

[Almuallim and Dietterich, 1991] Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *AAAI*, volume 91, pages 547–552. Citeseer.

[Alsinet et al., 2004] Alsinet, T., Manya, F., and Planes, J. (2004). A max-sat solver with lazy data structures. In *Ibero-American Conference on Artificial Intelligence*, pages 334–342. Springer.

[Argelich et al., 2018] Argelich, J., Li, C. M., Manyà, F., and Zhu, Z. (2018). Clause branching in maxsat and minsat.

[Argelich and Manya, 2007] Argelich, J. and Manya, F. (2007). Partial max-sat solvers with clause learning. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 28–40. Springer.

[Bai, 2005] Bai, R. (2005). *An investigation of novel approaches for optimising retail shelf space allocation*. PhD thesis, University of Nottingham.

[Bai et al., 2012] Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., and McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, 10(1) :43–66.

[Battiti et al., 1999] Battiti, R., Bertossi, A., and Cappelletti, A. (1999). Multilevel reactive tabu search for graph partitioning. *Preprint UTM*, 554.

[Biere et al., 1999] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). Symbolic model checking without bdds. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer.

[Biere et al., 2009] Biere, A., Heule, M., and van Maaren, H. (2009). *Handbook of satisfiability*, volume 185. IOS press.

[Blum and Rivest, 1989] Blum, A. and Rivest, R. L. (1989). Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501.

[Blum et al., 2008] Blum, C., Vallès, M. Y., and Blesa, M. J. (2008). An ant colony optimization algorithm for dna sequencing by hybridization. *Computers & Operations Research*, 35(11) :3620–3635.

[Bolón-Canedo et al., 2013] Bolón-Canedo, V., Sánchez-Maroño, N., and Alonso-Betanzos, A. (2013). A review of feature selection methods on synthetic data. *Knowledge and information systems*, 34(3) :483–519.

[Boughaci et al., 2008] Boughaci, D., Benhamou, B., and Drias, H. (2008). Scatter search and genetic algorithms for max-sat problems. *Journal of Mathematical Modelling and Algorithms*, 7(2) :101–124.

[Boughaci et al., 2004] Boughaci, D., Drias, H., and Benhamou, B. (2004). Solving max-sat problems using a memetic evolutionary meta-heuristic. In *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, volume 1, pages 480–484. IEEE.

[Bouhmala, 2012] Bouhmala, N. (2012). A multilevel approach applied to sat-encoded problems. *VLSI Design*, page 167.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and regression trees. *Wadsworth Int Group*, 37(15) :237–251.

[Burges, 1998] Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2) :121–167.

[Burke et al., 2013] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics : A survey of the state of the art. *Journal of the Operational Research Society*, 64(12) :1695–1724.

[Burke et al., 2010] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer.

[Burke et al., 2009] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*, pages 177–201. Springer.

[Burke et al., 2019] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2019). A classification of hyper-heuristic approaches : Revisited. In *Handbook of Metaheuristics*, pages 453–477. Springer.

[Burke et al., 2012] Burke, E. K., Kendall, G., Mısır, M., and Özcan, E. (2012). Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1) :73–90.

[Burke et al., 2003] Burke, E. K., Kendall, G., and Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of heuristics*, 9(6) :451–470.

[Cha et al., 1997] Cha, B., Iwama, K., Kambayashi, Y., and Miyazaki, S. (1997). Local search algorithms for partial maxsat. *AAAI/IAAI*, 263268.

[Chakhlevitch and Cowling, 2008] Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics : recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer.

[Chapelle and Li, 2011] Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257.

[Chebouba et al., 2018] Chebouba, L., Boughaci, D., and Guziolowski, C. (2018). Proteomics versus clinical data and stochastic local search based feature selection for acute myeloid leukemia patients' classification. *Journal of medical systems*, 42(7) :129.

[Chifu et al., 2018] Chifu, V. R., Pop, C. B., Birladeanu, A., Dragoi, N., and Salomie, I. (2018). Choice function-based constructive hyper-heuristic for generating personalized healthy menu recommendations. In *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 111–118. IEEE.

[Choong et al., 2017] Choong, S. S., Wong, L.-P., and Lim, C. P. (2017). An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 357–362. IEEE.

[Choong et al., 2019] Choong, S. S., Wong, L.-P., and Lim, C. P. (2019). An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. *Swarm and evolutionary computation*, 44 :622–635.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM.

[Cowling et al., 2000] Cowling, P., Kendall, G., and Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 176–190. Springer.

[Cowling et al., 2002] Cowling, P., Kendall, G., and Soubeiga, E. (2002). Hyperheuristics : A tool for rapid prototyping in scheduling and optimisation. In *Workshops on Applications of Evolutionary Computation*, pages 1–10. Springer.

[Crainic et al., 2006] Crainic, T. G., Li, Y., and Toulouse, M. (2006). A first multilevel cooperative algorithm for capacitated multicommodity network design. *Computers & operations research*, 33(9) :2602–2622.

[Dash and Liu, 1997] Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent data analysis*, 1(1-4) :131–156.

[Davis et al., 1962] Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397.

[Din et al., 2017] Din, F., Alsewari, A. R. A., and Zamli, K. Z. (2017). A parameter free choice function based hyper-heuristic strategy for pairwise test generation. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 85–91. IEEE.

[Dowsland et al., 2007] Dowsland, K. A., Soubeiga, E., and Burke, E. (2007). A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3) :759–774.

[Drake et al., 2015] Drake, J. H., Özcan, E., and Burke, E. K. (2015). Modified choice function heuristic selection for the multidimensional knapsack problem. In *Genetic and Evolutionary Computing*, pages 225–234. Springer.

[Dréo, 2004] Dréo, J. (2004). *Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical*. PhD thesis, PhD thesis, Paris 12.

[Emary et al., 2016] Emary, E., Zawbaa, H. M., and Hassanien, A. E. (2016). Binary ant lion approaches for feature selection. *Neurocomputing*, 213 :54–65.

[Fernandes and Lourenço, 2007] Fernandes, S. and Lourenço, H. (2007). Hybrids combining local search heuristics with exact algorithms. In *V Congreso Espanol sobre Metaheurısticas, Algoritmos Evolutivos y Bioinspirados*, pages 269–274.

[Fisher and Thompson, 1961] Fisher, H. and Thompson, G. (1961). Probabilistic learning combinations of local job-shop scheduling rules. In *in Factory Scheduling Conference*.

[Fisher and Thompson, 1963] Fisher, H. and Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251.

[Gardeux, 2011] Gardeux, V. (2011). *Conception d'heuristiques d'optimisation pour les problèmes de grande dimension : application à l'analyse de données de puces à ADN*. PhD thesis.

[Garrido and Riff, 2007a] Garrido, P. and Riff, M. C. (2007a). Collaboration between hyperheuristics to solve strip-packing problems. In *International Fuzzy Systems Association World Congress*, pages 698–707. Springer.

[Garrido and Riff, 2007b] Garrido, P. and Riff, M.-C. (2007b). An evolutionary hyperheuristic to solve strip-packing problems. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 406–415. Springer.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549.

[Gottlieb et al., 2002] Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary computation*, 10(1) :35–50.

[Graepel et al., 2010] Graepel, T., Candela, J. Q., Borchert, T., and Herbrich, R. (2010). Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress.

[Granmo, 2010] Granmo, O.-C. (2010). Solving two-armed bernoulli bandit problems using a bayesian learning automaton. *International Journal of Intelligent Computing and Cybernetics*, 3(2) :207–234.

[Guyon and Elisseeff, 2003] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar) :1157–1182.

[Hendrickson and Leland, 1995] Hendrickson, B. and Leland, R. W. (1995). A multi-level algorithm for partitioning graphs. *SC*, 95(28) :1–14.

[Heras et al., 2007] Heras, F., Larrosa, J., and Oliveras, A. (2007). Minimaxsat : A new weighted max-sat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 41–55. Springer.

[Hireche et al., 2020] Hireche, C., Drias, H., and Moulai, H. (2020). Grid based clustering for satisfiability solving. *Applied Soft Computing*, page 106069.

[Hoos and Stützle, 2000] Hoos, H. H. and Stützle, T. (2000). Local search algorithms for sat : An empirical evaluation. *Journal of Automated Reasoning*, 24(4) :421–481.

[Hsiao et al., 2012] Hsiao, P.-C., Chiang, T.-C., and Fu, L.-C. (2012). A vns-based hyper-heuristic with adaptive computational budget of local search. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

[Ishibuchi and Nakashima, 2000] Ishibuchi, H. and Nakashima, T. (2000). Multi-objective pattern and feature selection by a genetic algorithm. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 1069–1076. Morgan Kaufmann Publishers Inc.

[Jain and Zongker, 1997] Jain, A. and Zongker, D. (1997). Feature selection : Evaluation, application, and small sample performance. *IEEE transactions on pattern analysis and machine intelligence*, 19(2) :153–158.

[Jensen and Shen, 2003] Jensen, R. and Shen, Q. (2003). Finding rough set reducts with ant colony optimization. In *Proceedings of the 2003 UK workshop on computational intelligence*, volume 1, pages 15–22.

[Kabir et al., 2012] Kabir, M. M., Shahjahan, M., and Murase, K. (2012). A new hybrid ant colony optimization algorithm for feature selection. *Expert Systems with Applications*, 39(3) :3747–3763.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

[Karypis and Kumar, 1995] Karypis, G. and Kumar, V. (1995). Analysis of multilevel graph partitioning. In *Supercomputing'95 : Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, pages 29–29. IEEE.

[Kashef and Nezamabadi-pour, 2015] Kashef, S. and Nezamabadi-pour, H. (2015). An advanced aco algorithm for feature subset selection. *Neurocomputing*, 147 :271–279.

[Kaveh and Bondarabady, 2003] Kaveh, A. and Bondarabady, H. R. (2003). A hybrid graph-genetic method for domain decomposition. *Finite elements in analysis and design*, 39(13) :1237–1247.

[Kendall et al., 2002] Kendall, G., Cowling, P., and Soubeiga, E. (2002). Choice function and random hyperheuristics. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 667–671.

[Kendall and Hussin, 2004] Kendall, G. and Hussin, N. M. (2004). A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 270–293. Springer.

[Kendall and Mohamad, 2004] Kendall, G. and Mohamad, M. (2004). Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004)(IEEE Cat. No. 04EX955)*, volume 2, pages 769–773. IEEE.

[Kohavi and John, 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2) :273–324.

[Korošec and Šilc, 2012] Korošec, P. and Šilc, J. (2012). Using stigmergy to solve numerical optimization problems. *Computing and Informatics*, 27(3) :377–402.

[Korošec et al., 2004] Korošec, P., Šilc, J., and Robič, B. (2004). Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel computing*, 30(5-6) :785–801.

[Kroc et al., 2009] Kroc, L., Sabharwal, A., Gomes, C. P., and Selman, B. (2009). Integrating systematic and local search paradigms : A new strategy for maxsat. In *Twenty-First International Joint Conference on Artificial Intelligence*.

[Kuncheva and Jain, 1999] Kuncheva, L. I. and Jain, L. C. (1999). Nearest neighbor classifier : Simultaneous editing and feature selection. *Pattern recognition letters*, 20(11-13) :1149–1156.

[Langham and Grant, 1999] Langham, A. and Grant, P. (1999). A multilevel k–way partitioning algorithm for finite element meshes using competing ant colonies. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1602–1608. Morgan Kaufmann Publishers Inc.

[Lardeux et al., 2006] Lardeux, F., Saubion, F., and Hao, J.-K. (2006). Gasat : a genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2) :223–253.

[Larrosa et al., 2008] Larrosa, J., Heras, F., and De Givry, S. (2008). A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2-3) :204–233.

[Lassouaoui and Boughaci, 2014] Lassouaoui, M. and Boughaci, D. (2014). A choice function hyper-heuristic for the winner determination problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, pages 303–314. Springer.

[Layeb et al., 2010] Layeb, A., Deneche, A. H., and Meshoul, S. (2010). A new artificial immune system for solving the maximum satisfiability problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 136–142. Springer.

[Layeb and Saidouni, 2008] Layeb, A. and Saidouni, D.-E. (2008). A new quantum evolutionary local search algorithm for max 3-sat problem. In *International Workshop on Hybrid Artificial Intelligence Systems*, pages 172–179. Springer.

[Lee and Kim, 2015] Lee, J. and Kim, D.-W. (2015). Memetic feature selection algorithm for multi-label classification. *Information Sciences*, 293 :80–96.

[Lehrbaum and Musliu, 2012] Lehrbaum, A. and Musliu, N. (2012). A new hyperheuristic algorithm for cross-domain search problems. In *International Conference on Learning and Intelligent Optimization*, pages 437–442. Springer.

[Li et al., 2009] Li, C. M., Manya, F., Mohamedou, N., and Planes, J. (2009). Exploiting cycle structures in max-sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 467–480. Springer.

[Li et al., 2005] Li, C. M., Manya, F., and Planes, J. (2005). Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In *International conference on principles and practice of constraint programming*, pages 403–414. Springer.

[Li, 2004] Li, X. Y. (2004). *Optimization Algorithms for the Minimum-Cost Satisability Problem.* PhD thesis, North Carolina State University.

[Lin and Su, 2007] Lin, H. and Su, K. (2007). Exploiting inference rules to compute lower bounds for max-sat solving. In *IJCAI*, volume 7, pages 2334–2339.

[Lin et al., 2008] Lin, S.-W., Ying, K.-C., Chen, S.-C., and Lee, Z.-J. (2008). Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4) :1817–1824.

[Liu et al., 2009] Liu, H., Sun, J., Liu, L., and Zhang, H. (2009). Feature selection with dynamic mutual information. *Pattern Recognition*, 42(7) :1330–1339.

[Liu and De Melo, 2017] Liu, S. and De Melo, G. (2017). Should algorithms for random sat and max-sat be different ? In *Thirty-First AAAI Conference on Artificial Intelligence*.

[Luo et al., 2014] Luo, C., Cai, S., Wu, W., Jie, Z., and Su, K. (2014). Ccls : an efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7) :1830–1843.

[Maashi et al., 2015] Maashi, M., Kendall, G., and Özcan, E. (2015). Choice function based hyper-heuristics for multi-objective optimization. *Applied Soft Computing*, 28 :312–326.

[Mafarja and Abdullah, 2013] Mafarja, M. and Abdullah, S. (2013). Investigating memetic algorithm in solving rough set attribute reduction. *International Journal of Computer Applications in Technology*, 48(3) :195–202.

[Marchiori and Rossi, 1999] Marchiori, E. and Rossi, C. (1999). A flipping genetic algorithm for hard 3-sat problems. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 393–400. Morgan Kaufmann Publishers Inc.

[Marill and Green, 1963] Marill, T. and Green, D. (1963). On the effectiveness of receptors in recognition systems. *IEEE transactions on Information Theory*, 9(1) :11–17.

[Marques-Silva and Sakallah, 1999] Marques-Silva, J. P. and Sakallah, K. A. (1999). Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521.

[Mastrolilli and Gambardella, 2005] Mastrolilli, M. and Gambardella, L. M. (2005). Maximum satisfiability : how good are tabu search and plateau moves in the worst-case ? *European Journal of Operational Research*, 166(1) :63–76.

[May and Leslie, 2011] May, B. C. and Leslie, D. S. (2011). Simulation studies in optimistic bayesian sampling in contextual-bandit problems. *Statistics Group, Department of Mathematics, University of Bristol*, 11 :02.

[Meignan, 2008] Meignan, D. (2008). *Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques, Application aux problèmes d'optimisation de réseaux de transports*. PhD thesis.

[Menaï and Batouche, 2005] Menaï, M. E. B. and Batouche, M. (2005). A backbone-based co-evolutionary heuristic for partial max-sat. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 155–166. Springer.

[Miyazaki, 1996] Miyazaki, S. (1996). Database queries as combinatorial optimization problems. In *Proc. International Symposium on Cooperative Database Systems for Advanced Applications*, pages 448–454.

[Montazeri, 2016] Montazeri, M. (2016). Hhfs : Hyper-heuristic feature selection. *Intelligent Data Analysis*, 20(4) :953–974.

[Moradi and Rostami, 2015] Moradi, P. and Rostami, M. (2015). Integration of graph clustering with ant colony optimization for feature selection. *Knowledge-Based Systems*, 84 :144–161.

[Narendra and Fukunaga, 1977] Narendra, P. M. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers*, (9) :917–922.

[Nekkaa and Boughaci, 2015] Nekkaa, M. and Boughaci, D. (2015). A memetic algorithm with support vector machine for feature selection and classification. *Memetic Computing*, 7(1) :59–73.

[Nekkaa and Boughaci, 2016] Nekkaa, M. and Boughaci, D. (2016). Hybrid harmony search combined with stochastic local search for feature selection. *Neural Processing Letters*, 44(1) :199–220.

[Nilsson, 2003] Nilsson, C. (2003). Heuristics for the traveling salesman problem. *Linkoping University*, 38 :00085–9.

[Ochoa et al., 2009] Ochoa, G., Vázquez-Rodríguez, J. A., Petrovic, S., and Burke, E. (2009). Dispatching rules for production scheduling : a hyper-heuristic landscape analysis. In *2009 IEEE congress on evolutionary computation*, pages 1873–1880. IEEE.

[Oduntan, 2006] Oduntan, I. O. (2006). A multilevel search algorithm for feature selection in biomedical data.

[Özcan et al., 2009] Özcan, E., Bykov, Y., Birben, M., and Burke, E. K. (2009). Examination timetabling using late acceptance hyper-heuristics. In *2009 IEEE Congress on Evolutionary Computation*, pages 997–1004. IEEE.

[Özcan et al., 2012] Özcan, E., Misir, M., Ochoa, G., and Burke, E. K. (2012). A reinforcement learning : great-deluge hyper-heuristic for examination timetabling. In *Modeling, Analysis, and Applications in Metaheuristic Computing : Advancements and Trends*, pages 34–55. IGI Global.

[Palanisamy and Kanmani, 2012] Palanisamy, S. and Kanmani, S. (2012). Artificial bee colony approach for optimizing feature selection. *International Journal of Computer Science Issues (IJCSI)*, 9(3) :432.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). Computational complexity.

[Pillay, 2008] Pillay, N. (2008). An analysis of representations for hyper-heuristics for the uncapacitated examination timetabling problem in a genetic programming system. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries : riding the wave of technology*, pages 188–192. ACM.

[Pour et al., 2018] Pour, S. M., Drake, J. H., and Burke, E. K. (2018). A choice function hyper-heuristic framework for the allocation of maintenance tasks in danish railways. *Computers & Operations Research*, 93 :15–26.

[Pudil et al., 1994a] Pudil, P., Ferri, F. J., Novovicova, J., and Kittler, J. (1994a). Floating search methods for feature selection with nonmonotonic criterion functions. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C : Signal Processing (Cat. No. 94CH3440-5)*, volume 2, pages 279–283. IEEE.

[Pudil et al., 1994b] Pudil, P., Novovičová, J., and Kittler, J. (1994b). Floating search methods in feature selection. *Pattern recognition letters*, 15(11) :1119–1125.

[Ramírez and Geffner, 2007] Ramírez, M. and Geffner, H. (2007). Structural relaxations by variable renaming and their compilation for solving mincost-sat. In *International conference on principles and practice of constraint programming*, pages 605–619. Springer.

[Rintanen et al., 2006] Rintanen, J., Heljanko, K., and Niemelä, I. (2006). Planning as satisfiability : parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13) :1031–1080.

[Rodney et al., 2005] Rodney, D., Soper, A., and Walshaw, C. (2005). Multilevel refinement for the vehicle routing problem. In *Proceedings of PlanSIG 2005, 24th Annual Workshop of UK Planning & Scheduling Special Interest Group*, pages 96–97. PlanSIG.

[Rodney et al., 2008] Rodney, D., Soper, A. J., and Walshaw, C. (2008). Multilevel approaches applied to the capacitated clustering problem. In *CSC*, pages 271–277.

[Rodríguez et al., 2007a] Rodríguez, J. A. V., Petrovic, S., and Salhi, A. (2007a). An investigation of hyper-heuristic search spaces. In *2007 IEEE Congress on Evolutionary Computation*, pages 3776–3783. IEEE.

[Rodríguez et al., 2007b] Rodríguez, J. V., Petrovic, S., and Salhi, A. (2007b). A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications. MISTA : Paris, France*, pages 506–513.

[Romem et al., 1995] Romem, Y., Rudolph, L., and Stein, J. (1995). Adapting multilevel simulated annealing for mapping dynamic irregular problems. In *Intl Parallel Processing Symp*, pages 65–72.

[Sandholm, 2002] Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial intelligence*, 135(1-2) :1–54.

[Schapire, 1999] Schapire, R. E. (1999). A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406.

[Schiezaro and Pedrini, 2013] Schiezaro, M. and Pedrini, H. (2013). Data feature selection based on artificial bee colony algorithm. *EURASIP Journal on Image and Video Processing*, 2013(1) :47.

[Scott, 2010] Scott, S. L. (2010). A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6) :639–658.

[Selman et al., 1994] Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *AAAI*, volume 94, pages 337–343.

[Selman et al., 1992] Selman, B., Levesque, H. J., Mitchell, D. G., et al. (1992). A new method for solving hard satisfiability problems. In *Aaai*, volume 92, pages 440–446. Citeseer.

[Shanthi and Bhaskaran, 2014] Shanthi, S. and Bhaskaran, V. M. (2014). Modified artificial bee colony based feature selection : a new method in the application of mammogram image classification. *Int. J. Sci. Eng. Technol. Res*, 3(6) :1664–1667.

[Sin, 2011] Sin, E. S. (2011). Reinforcement learning with egd based hyper heuristic system for exam timetabling problem. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 462–466. IEEE.

[Sirenko, 2009] Sirenko, S. (2009). Classification of heuristic methods in combinatorial optimization. *Information Theories & Applications*, pages 303–322.

[Smith et al., 2005] Smith, A., Veneris, A., Ali, M. F., and Viglas, A. (2005). Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10) :1606–1621.

[Smyth et al., 2003] Smyth, K., Hoos, H. H., and Stützle, T. (2003). Iterated robust tabu search for max-sat. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 129–144. Springer.

[Soper et al., 2004] Soper, A. J., Walshaw, C., and Cross, M. (2004). A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *Journal of Global Optimization*, 29(2) :225–241.

[Soubeiga, 2003] Soubeiga, E. (2003). *Development and application of hyper-heuristics to personnel scheduling*. PhD thesis, University of Nottingham.

[Sun and Zhang, 2010] Sun, D. and Zhang, D. (2010). Bagging constraint score for feature selection with pairwise constraints. *Pattern Recognition*, 43(6) :2106–2118.

[Sun et al., 2012] Sun, X., Liu, Y., Li, J., Zhu, J., Chen, H., and Liu, X. (2012). Feature evaluation and selection with cooperative game theory. *Pattern recognition*, 45(8) :2992–3002.

[Tahir and Smith, 2010] Tahir, M. A. and Smith, J. (2010). Creating diverse nearest-neighbour ensembles using simultaneous metaheuristic feature selection. *Pattern Recognition Letters*, 31(11) :1470–1480.

[Talbi, 2009] Talbi, E.-G. (2009). *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons.

[Tang et al., 2013] Tang, L., Rosales, R., Singh, A., and Agarwal, D. (2013). Automatic ad format selection via contextual bandits. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1587–1594. ACM.

[Teng, 1999] Teng, S.-H. (1999). Coarsening, sampling, and smoothing : Elements of the multilevel method. In *Algorithms for Parallel Processing*, pages 247–276. Springer.

[Thompson, 1933] Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4) :285–294.

[Toulouse et al., 1999] Toulouse, M., Thulasiraman, K., and Glover, F. (1999). Multi-level cooperative search : A new paradigm for combinatorial optimization and an application to graph partitioning. In *European Conference on Parallel Processing*, pages 533–542. Springer.

[Tsai et al., 2013] Tsai, C.-F., Eberle, W., and Chu, C.-Y. (2013). Genetic algorithms in feature and instance selection. *Knowledge-Based Systems*, 39 :240–247.

[Unler and Murat, 2010] Unler, A. and Murat, A. (2010). A discrete particle swarm optimization method for feature selection in binary classification problems. *European Journal of Operational Research*, 206(3) :528–539.

[Walshaw, 2001a] Walshaw, C. (2001a). *A multilevel approach to the graph colouring problem*. Citeseer.

[Walshaw, 2001b] Walshaw, C. (2001b). *A multilevel lin-kernighan-helsgaun algorithm for the travelling salesman problem*. Citeseer.

[Walshaw, 2002] Walshaw, C. (2002). A multilevel approach to the travelling salesman problem. *Operations Research*, pages 862–877.

[Walshaw, 2008] Walshaw, C. (2008). Multilevel refinement for combinatorial optimisation : Boosting metaheuristic performance. In *Hybrid Metaheuristics*, pages 261–289. Springer.

[Walshaw and Cross, 2000] Walshaw, C. and Cross, M. (2000). Mesh partitioning : a multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1) :63–80.

[Wang et al., 2017] Wang, H., Jing, X., and Niu, B. (2017). A discrete bacterial algorithm for feature selection in classification of microarray gene expression cancer data. *Knowledge-Based Systems*, 126 :8–19.

[Wang et al., 2007] Wang, J., Hedar, A.-R., and Wang, S. (2007). Scatter search for rough set attribute reduction. In *2007 Second International Conference on Bio-Inspired Computing : Theories and Applications*, pages 236–240. IEEE.

[Whitney, 1971] Whitney, A. W. (1971). A direct method of nonparametric measurement selection. *IEEE Transactions on Computers*, 100(9) :1100–1103.

[Xu et al., 2003] Xu, H., Rutenbar, R. A., and Sakallah, K. (2003). sub-sat : a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6) :814–820.

[Xu et al., 2019] Xu, Z., He, K., and Li, C.-M. (2019). An iterative path-breaking approach with mutation and restart strategies for the max-sat problem. *Computers & Operations Research*, 104 :49–58.

[Yang et al., 2008] Yang, C.-S., Chuang, L.-Y., Chen, Y.-J., and Yang, C.-H. (2008). Feature selection using memetic algorithms. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, volume 1, pages 416–423. IEEE.

[Zamli et al., 2016] Zamli, K. Z., Alkazemi, B. Y., and Kendall, G. (2016). A tabu search hyper-heuristic strategy for t-way test suite generation. *Applied Soft Computing*, 44 :57–74.

[Zawbaa et al., 2015] Zawbaa, H. M., Emary, E., and Parv, B. (2015). Feature selection based on antlion optimization algorithm. In *2015 Third World Conference on Complex Systems (WCCS)*, pages 1–7. IEEE.

[Zhu et al., 2007] Zhu, Z., Ong, Y.-S., and Dash, M. (2007). Wrapper–filter feature selection algorithm using a memetic framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1) :70–76.