Métaheuristiques hybrides distribuées et massivement parallèles

Omar Abdelkafi

Université de Haute-Alsace

École Doctorale de Mathématiques, Sciences de l'Information et de l'Ingénieur (ED 269)

THESE DE DOCTORAT
EN INFORMATIQUE

Par

Omar ABDELKAFI

Sujet de la thèse :

Métaheuristiques hybrides distribuées et massivement parallèles

Thèse soutenue le 07 Novembre 2016

Laetitia JOURDAN	Professeur des Universités	Université de Lille 1	Présidente
Abderrafiâa KOUKAM	Professeur des Universités	Université de Technologie de Belfort-Montbéliard	Rapporteur
Ammar OULAMARA	Professeur des Universités	Université de Lorraine	Rapporteur
René SCHOTT	Professeur des Universités	Université de Lorraine	Examinateur
Jean-Louis PAILLAUD	Directeur de Recherche CNRS	Université de Haute-Alsace	Examinateur
Julien LEPAGNOT	Maitre de conférences	Université de Haute-Alsace	Co-encadrant
Lhassane IDOUMGHAR	Professeur des Universités	Université de Haute-Alsace	Directeur de thèse

Liste des publications

Revues internationales avec comité de lecture :

[1] Omar Abdelkafi, Lhassane Idoumghar and Julien Lepagnot, *A Survey on the Metaheuristics applied to QAP for the Graphics Processing Units*, Parallel Processing Letters, 26(3), pp. 1-20, 2016.

Actes de conférences internationales avec comité de lecture :

[2] Omar Abdelkafi, Lhassane Idoumghar, Julien Lepagnot and Mathieu Brévilliers, *Data exchange topologies for the DISCO-HITS algorithm to solve the QAP*, 2016 International Conference on Swarm Intelligence Based Optimization, ICSIBO'2016, June 13-14, Mulhouse, France**,** pp. 30-37, 2016.

[3] Mathieu Brévilliers, Omar Abdelkafi, Julien Lepagnot and Lhassane Idoumghar*, Fast Hybrid BSA-DE-SA Algorithm on GPU*, 2016 International Conference on Swarm Intelligence Based Optimization, ICSIBO'2016, June 13-14, Mulhouse, France, pp. 46-53, 2016.

[4] Omar Abdelkafi, Lhassane Idoumghar, Julien Lepagnot and Mathieu Brévilliers, *A GPU-based parallel neighborhood evaluation for ITSSD,* 12th Biennal International Conference on Artificial Evolution (EA 2015), Lyon, October 26-28, pp. 327-324, 2015.

[5] Mathieu Brévilliers, Omar Abdelkafi, Julien Lepagnot and Lhassane Idoumghar, *Idol-Guided Backtracking Search Optimization Algorithm,* 12th Biennal International Conference on Artificial Evolution (EA 2015), Lyon, October 26-28, pp. 277-284, 2015.

[6] Omar Abdelkafi, Lhassane Idoumghar and Julien Lepagnot, *Distributed Multistart Hybrid Iterative Tabu Search*, IEEE International Conference on Systems, Man, and Cybernetics (SMC 2015): Big Data Analytics for Human-Centric Systems, Hong Kong, October 9-12, pp. 1962-1967, 2015.

[7] Omar Abdelkafi, Lhassane Idoumghar and Julien Lepagnot, *Comparison of Two Diversification Methods to Solve the Quadratic Assignment Problem,* International Conference On Computational Science (ICCS 2015): Computational Science at the Gates of Nature, Volume 51, in Reykjavik, Iceland, June 1-3, pp. 2703–2707, 2015.

[8] Omar Abdelkafi, Julien Lepagnot and Lhassane Idoumghar, *Multi-level parallelization for hybrid ACO*, 2014 International Conference on Swarm Intelligence Based Optimization, ICSIBO'2014, May 13-14, Mulhouse, France, Revised selected paper, LNCS 8472, pp. 60–67, 2014.

Liste des abréviations

A

AG : Algorithme Génétique

AE : Algorithme Evolutionnaire

ACO : Ant Colony Optimization

AS : Ant System

AU : Asymmetric Unit

Al : Aluminium

B

B&B : Branch and Bound

BKS : Best Known Solution

BLS : Breakout Local Search

BMA : Breakout Memetic Algorithm

BS : Bushuev-Sastre

C

CPU : Central Processing Unit

CUDA : Compute Unified Device Architecture

cAS : cunning Ant System

c-ant : cunning ant

CPTS : Cooperative Parallel Tabu Search

CBU : Composite Building Units

D

d-ant : donor ant

D-HITS : Distributed Hybrid Iterative Tabu Search

DG : Diversification de Glover

DISCO-HITS : DIStance COoperation Hybrid Iterative Tabu Search

DM-HITS : Distributed Multi-start Hybrid Iterative Tabu Search

F

FANT : Fast Ant

FI : Fourmi Intelligente

G

GPU : Graphic Processing Unit

GPGPU : General-Purpose Computing on Graphics Processing Units

GULP : General Utility Lattice Program

H

HITS : Hybrid Iterative Tabu Search

I

ITS : Iterative Tabu Search

IHGA : Improved Hybrid Genetic Algorithm

L

LJC : Lennard Jones Cluster

M

MPI : Message Passing Interface

MATA : Move-Cost Thread Assignment

MMAS : Max-Min Ant System

MR : Membered Ring

MFI : Mobil-5 FIve

MEGA-HZ : MEmory Genetic Algorithme Hybridized for Zeolite

N

NOW : Network Of Workstations

O

OpenCL : Open Computing Language

OCF : Optimisation par Colonies de Fourmis

OEP : Optimisation par Essaims de Particules

OSDA : Organic Structure Directing Agent

OPX : One Point Crossover

O : Oxygène

P

P-métaheuristiques : Population metaheuristics

PILS : Population-based Iterated Local Search

P-GHAZ : Parallel Genetic Hybrid Algorithm for Zeolite

Q

QAP : Quadratic Assignment Problem

R

RL : Recherche Locale

RLI : Recherche Locale Itérative

RT : Recherche Taboue

RTI : Recherche Taboue Itérative

RS : Recuit Simulé

Ro-TS : Robust Tabu Search

RX : Random Crossover

S

S-métaheuristiques : Single metaheuristics

SIMD : Single Instruction Multiple Data

SPMD : Single Program Multiple Data

SC-IZA : Structure Commission of the International Zeolite Association

Si : Silicium

T

TSP : Traveling Salesman Problem

U

UAL : Unités Arithmétiques et Logiques

UC : Unité de contrôle (chapitre 1)

UC : Unité Cellulaire (chapitre 4)

UX : Uniform crossover

UTL : mulhoUse-TwLve

UFF : Universal Force Field

Z

ZSM-5 : Zeolite Socony Mobil-5

ZSP : Zeolites Structure Problem

Table des matières

Table des figures

Liste des algorithmes

Liste des tableaux

Introduction générale

Les travaux menés dans plusieurs domaines scientifiques et secteurs industriels comprennent la résolution de problèmes d'optimisation. Ces problèmes peuvent intervenir dans différentes disciplines telles que la logistique, le transport, la chimie ou l'énergie. Dans ce cadre, les chercheurs ont tentés de résoudre ces problèmes en exploitant la puissance de calcul des ordinateurs. Les approches qui ont eu le plus grand succès sont basées sur des métaheuristiques. Ces métaheuristiques, appelées aussi les heuristiques génériques, sont des algorithmes approximatifs qui sont capables de trouver une solution satisfaisante pour des problèmes complexes en un temps raisonnable.

La majorité des métaheuristiques publiées dans la littérature sont d'une façon ou d'une autre hybrides. Le théorème du *no free lunch* (Wolpert & Macready, 1997) indique qu'une métaheuristique ne peut prétendre être plus efficace qu'une autre sur tous les problèmes possibles. Néanmoins, en l'implémentant et en la paramétrant d'une certaine façon, elle peut être plus adaptée à certaines classes de problèmes. Ainsi, il est souvent nécessaire d'adapter une métaheuristique aux problèmes traités en ajustant, entre autres, ses capacités d'*intensification* et de *diversification*. L'intensification représente une phase d'exploration en profondeur qui consiste à appliquer la même transformation élémentaire afin d'obtenir le maximum de voisins. La diversification consiste à appliquer diverses transformations élémentaires afin d'obtenir le maximum de voisins avec différentes transformations (Teghem & Pirlot, 2002). La spécificité de l'hybridation des métaheuristiques est qu'elle est capable de donner à l'algorithme la capacité d'explorer l'espace de re-

cherche avec plusieurs techniques différentes. Toutefois, cette hybridation a forcément un coût. D'une manière générale, les métaheuristiques hybrides sont gourmandes en temps de calcul. Afin de remédier à ce problème, une des solutions qui peut être retenue est la parallélisation. Les métaheuristiques parallèles ont été adoptées par les chercheurs vus leurs apports importants dans l'accélération de la recherche, l'amélioration de la qualité de la solution obtenue, l'amélioration de la robustesse et la capacité à résoudre des problèmes de très grandes tailles (Talbi, 2009).

Pour exécuter des algorithmes parallèles, des plateformes adaptées sont nécessaires, telles que les supercalculateurs, les clusters, les GPU (*Graphic Processing Unit*), les FPGA (*field-Programmable Gate Array*), etc. Deux plateformes sont particulièrement intéressantes, qui sont les clusters et les GPU. Les clusters sont faciles à mettre en place grâce au grand nombre d'ordinateurs qui se trouvent dans nos structures (universités, usines, entreprises, etc.). En effet, pour avoir un cluster, il suffit d'avoir un ensemble de machines reliées entre elles via un réseau, comme dans le cas d'une salle de travaux pratiques à l'université. Contrairement à des plateformes qui coûtent chères et qui sont très difficiles d'accès, comme les supercalculateurs, les clusters sont accessibles et peuvent utiliser une infrastructure déjà existante.

La deuxième plateforme qui est intéressante est le GPU. Ces dernières années, l'évolution du GPU est beaucoup plus rapide que celle du CPU (*Central Processing Unit*). Cela est du, entre autres, à l'expansion de l'industrie des jeux vidéo et de sa demande gourmande en puissance graphique. En contrepartie, le CPU s'approche de plus en plus de ses limites physiques ce qui freine son évolution. L'utilisation du GPU dans des applications non-graphiques a eu un intérêt croissant dans le monde de l'optimisation combinatoire depuis les années 2009. Le succès du GPU revient en partie à son coût faible, car c'est une architecture grand public produite en série et facilement accessible. L'enjeu financier rend cette technologie très adéquate à l'implémentation des métaheuristiques parallèles pour la

résolution des problèmes combinatoires. L'objectif est d'adapter chaque métaheuristique à la plateforme GPU afin de tirer profit de ses avantages.

La parallélisation et l'hybridation pour les métaheuristiques sont deux concepts qui sont fortement liés. Dans plusieurs cas, faire une parallélisation implique de faire une hybridation. Ceci peut se manifester dans l'utilisation des heuristiques pour l'échange de données ou bien dans la parallélisation de plusieurs populations d'individus, i.e. de solutions, de départ. Les principaux travaux de la littérature dans le parallélisme ont pour objectif d'accélérer des algorithmes grâce à la puissance de calcul des architectures parallèles. Cette accélération est obtenue sans considérer un autre aspect important offert par la parallélisation, qui est la compétitivité sur la qualité des solutions. De surcroît, l'accélération est toujours relative au matériel utilisé alors que la qualité des solutions trouvées par différents algorithmes de la littérature est plus facilement comparable.

D'un autre côté, plusieurs frameworks de métaheuristiques (Cahon et al., 2004; Roberge et al., 2015) proposent des algorithmes hybrides et parallèles génériques. D'une manière globale, l'hybridation dans ces frameworks est destinée à plusieurs problèmes combinatoires, ce qui augmente le niveau de généricité et d'abstraction du code et donc, rend leur utilisation plus difficile et complique l'analyse de l'hybridation et de la parallélisation.

En prenant en compte ces éléments, notre contribution s'oriente vers l'implémentation de différentes métaheuristiques hybrides destinées à un problème particulier. Cela signifie que les hybridations proposées sont spécifiques aux problèmes traités. Du côté de la parallélisation, notre première priorité est l'amélioration de la qualité des solutions. L'objectif est de proposer des métaheuristiques hybrides distribuées sur les clusters et massivement parallèles sur le GPU pour traiter des problèmes d'optimisation. Ainsi, les propositions réalisées dans notre travail doivent, d'un côté être spécifiques à un problème donné et, d'un autre côté, privilégier la qualité des solutions obtenues. Ces métaheuristiques propo-

sées entrent dans l'optique de réaliser une bibliothèque de métaheuristiques hybrides et parallèles pour résoudre des problèmes particuliers.

À l'issue de ce travail, cinq niveaux de parallélisation et deux niveaux d'hybridation sont utilisés. L'impact de ces niveaux est étudié à travers trois problèmes d'optimisation, à savoir, le problème du voyageur de commerce, le problème d'affectation quadratique et le problème de la résolution structurale des zéolithes.

Les principales contributions de cette thèse consistent à :

- Mettre en place les niveaux de parallélisation et d'hybridation à travers le problème du voyageur de commerce.
- Expérimenter et analyser l'impact de plusieurs niveaux proposés sur le problème d'affectation quadratique.
- Modéliser et résoudre le problème de la résolution structurale des zéolithes en utilisant les résultats obtenus à travers les deux premières contributions.

Ce présent mémoire est rédigé en quatre chapitres. Dans le premier chapitre, nous présentons une vue globale sur l'hybridation et sur la parallélisation des métaheuristiques. Nous évoquons dans chaque partie les principaux travaux de la littérature qui s'intéressent aux métaheuristiques hybrides, dans un premier temps, puis aux métaheuristiques parallèles dans un second temps. Ensuite, nous formulons les objectifs et les motivations de ce travail.

Dans le deuxième chapitre, nous présentons tous les concepts de base qui permettent de développer des métaheuristiques hybrides, distribuées et massivement parallèles. Ces concepts sont appliqués à travers une métaheuristique hybride et parallèle d'optimisation par les colonies de fourmis sur le problème du voyageur de commerce. Cet algorithme utilise les cinq niveaux de parallélisation ainsi que les deux niveaux d'hybridation définis dans la première partie de ce même chapitre.

Dans le troisième chapitre, nous proposons quatre variantes d'algorithmes basées sur une recherche taboue itérative avec des diversifications adaptatives pour résoudre le pro-

blème d'affectation quadratique. Les quatre variantes se distinguent par les protocoles d'échange d'informations entre les processus parallèles et les méthodes de diversification. Plusieurs niveaux d'hybridation et de parallélisation sont utilisés. Une analyse expérimentale est effectuée afin de pouvoir examiner l'impact des différents niveaux.

Le dernier chapitre est dédié au problème de la résolution structurale des zéolithes. Nous proposons deux nouvelles formulations de la fonction objectif pour évaluer le potentiel de la structure zéolitique trouvée. Deux métaheuristiques hybrides et parallèles sont également développées en utilisant les niveaux de parallélisation et d'hybridation adéquats. Une expérimentation pour analyser les résultats est également effectuée. Des analyses numériques et chimiques sont réalisées pour valoriser les résultats obtenus.

Enfin, dans la conclusion générale du manuscrit, nous récapitulons nos contributions et nous proposons des perspectives, sur la base des travaux effectués.

Chapitre 1

État de l'art et contexte général de l'étude

Dans ce premier chapitre nous présentons les métaheuristiques hybrides et les différents aspects de parallélisation. D'abord nous présentons une généralisation des métaheuristiques hybrides ainsi qu'une revue de la littérature d'une sélection de ces méthodes. Ensuite, nous décrivons les métaheuristiques distribuées et massivement parallèles. Dans cette description, nous proposons un état de l'art sur une sélection d'entre elles. Finalement, nous présentons les objectifs et les motivations de ce travail.

1.1 Les métaheuristiques hybrides

1.1.1 Les métaheuristiques

Les métaheuristiques (Blum & Roli, 2003) sont des algorithmes heuristiques performants capables de résoudre, entre autres, les problèmes d'optimisation combinatoire. La caractéristique commune des métaheuristiques consiste à utiliser un mécanisme aléatoire d'exploration de l'espace des solutions. Cette caractéristique permet de déterminer ou

d'approcher un optimum global (Teghem & Pirlot, 2002). Contrairement aux algorithmes exacts (Puchinger & Raidl, 2005), les métaheuristiques ne garantissent pas la solution optimale. Elles proposent plutôt une solution acceptable dans un temps de calcul raisonnable pour des problèmes complexes (Siarry, 2014).

On trouve deux types de métaheuristiques de base (Talbi, 2009) :

- les métaheuristiques à base d'une seule solution (**les S-métaheuristiques**). Elles consistent à chercher les voisins à partir d'une solution de base pour obtenir de meilleures solutions. A partir d'une solution initiale, le principe est de passer d'une solution à une autre voisine d'une manière itérative pour améliorer la solution de base. Un des exemples de cette catégorie est la recherche taboue (Glover, 1990).
- les métaheuristiques à base d'une population de solutions (**les P-métaheuristiques**). Elles manipulent non pas une seul solution, mais un ensemble de solutions à chaque itération. L'objectif est d'améliorer les solutions d'une itération à une autre d'une manière itérative. Un des exemples de cette catégorie est l'algorithme génétique (Holland, 1975).

1.1.2 L'hybridation

Les métaheuristiques hybrides (Aouad et al., 2010; Bluma et al., 2011; Lepagnot et al., 2013; Brévilliers et al., 2015) sont considérées parmi les algorithmes les plus performants. Le principe est de combiner des métaheuristiques avec d'autres techniques d'optimisation. Ces techniques peuvent être d'autres métaheuristiques, des heuristiques, des méthodes exactes, etc.

Un grand nombre des métaheuristiques proposées dans la littérature sont hybrides (Bluma et al., 2011). En général, les métaheuristiques hybrides offrent un meilleur équilibre entre intensification et diversification. Dans la littérature, deux niveaux d'hybridation sont proposés (Talbi, 2009; Raidl et al., 2010).

- **L'hybridation à bas niveau** : elle consiste à embarquer ou à remplacer un mécanisme interne d'une métaheuristique par une autre méthode. En général, cette hybridation a pour but d'utiliser les propriétés de diversification de la métaheuristique de base et d'intensifier la recherche avec l'hybridation embarquée. Par exemple, ce niveau peut être réalisé en remplaçant la mutation d'un algorithme génétique par une recherche locale (Aarts & Lenstra, 1997).
- **L'hybridation à haut niveau** : elle consiste à combiner une métaheuristique avec d'autres méthodes sans que leur fonctionnements interne ne soient en relation. Le but est typiquement d'exécuter une séquence de métaheuristiques. Un exemple d'hybridation à haut niveau consiste à utiliser une P-metaheuristique pour proposer des solutions initiales pour une S-métaheuristique. Le modèle multi-colonies dans l'optimisation des colonies de fourmis (Dorigo & Stützle, 2004) et le modèle en ilôts dans l'algorithme génétique sont considérés comme des hybridations à haut niveau.

1.1.3 Revue de la littérature des métaheuristiques hybrides

La majorité des métaheuristiques récentes proposées dans la littérature peuvent être considérées comme des métaheuristiques hybrides. Elles ne suivent pas le paradigme des métaheuristiques traditionnelles. Le développement des métaheuristiques hybrides demande une bonne maitrise de différentes méthodes complémentaires afin de donner à l'algorithme plus de chances de trouver une meilleure solution. Blum et al. ont publié le premier livre dédié exclusivement aux métaheuristiques hybrides pour résoudre les problèmes d'optimisation combinatoire.

L'hybridation la plus naturelle dans la littérature est l'utilisation d'une S-métaheuristique embarquée dans une P-métaheuristique. Les S-metaheuristiques sont efficasses en intensification alors que les P-métaheuristiques ont une bonne capacité d'exploration (Teghem & Pirlot, 2002). Cette complémentarité entre les P-metaheuristiques et les S-métaheuristiques

permet une recherche plus efficace. C'est une hybridation à bas niveau. Un exemple de ces algorithmes est la proposition de Stützle & Hoos (1997) qui utilise l'Optimisation par Colonies de Fourmis (OCF) afin de positionner les fourmis dans des régions prometteuses de l'espace de recherche. Par la suite, une Recherche Locale (RL) s'occupe d'intensifier la recherche pour aboutir aux meilleures solutions. Il y'a également la proposition de Merz & Freisleben (2000). Cette approche consiste à hybrider un Algorithme Génétique (AG) avec une RL 2-opt. On peut citer également le travail de Idoumghar et al. (2001) qui consiste à remplacer la mutation d'un AG par un algorithme glouton dans un premier temps puis par une Recherche Taboue (RT) probabiliste. En 2008, différents AG sont testés et comparés par Drezner (2008). La meilleure variante de ce travail est celle qui combine un AG avec une RT. Il est courant d'appeler la fusion entre l'AG et la RL, algorithme mémétique (Krasnogor & Smith, 2005). En 2011, un nouvel algorithme évolutionnaire hybride qui combine entre l'Optimisation par Essaims de Particules (OEP) et le Recuit Simulé (RS) est proposé par Idoumghar et al. (2011). Cet algorithme permet à l'OEP d'éviter la convergence prématurée. Il y'a d'autres hybridations de bas niveau qui sont utilisées plus rarement dans la littérature. C'est l'utilisation de différentes P-métaheuristiques dans la même approche. Un exemple de cette approche est la proposition de Gong & Ruan (2004) qui hybride un AG avec l'OCF. Chaque individu de la population représente un chromosome et une fourmi en même temps. L'algorithme évolue comme un AG en exécutant des opérations de croisement et de mutation. Le point du croisement ou de la mutation est défini par une matrice de phéromones. La matrice de phéromones définie des segments avec une liaison forte (ce qui veut dire avec une grande quantité de phéromones entre les différents éléments du segment) pour éviter leur destruction par les opérateurs génétiques. Grâce à la matrice de phéromone, ces segments obtiennent une grande probabilité qui leur permet d'éviter le croisement et de rester intacte. De cette manière, les points de croisement et de mutation évitent les segments définis comme forts par la matrice de phéromones.

Le deuxième type d'hybridation est l'hybridation à haut niveau. Le principe est typiquement de faire se succéder des métaheuristiques différentes ou identiques. Chaque métaheuristique fournit une solution ou une population de départ pour la métaheuristique suivante. Afin d'échapper aux potentiels optima locaux, une ou plusieurs perturbations sont appliquées sur la nouvelle solution ou population de départ. Parmi les exemples d'hybridation à haut niveau, on trouve les algorithmes itératifs tels que la RTI (Misevicius, 2005; James et al., 2009b) ou la Recherche Locale Itérative (RLI) (Stützle, 2006). Leur principe consiste à exécuter successivement des recherches locales ou taboues avec différentes diversifications à chaque nouvelle recherche pour éviter la stagnation. Dans le même contexte, il y'a la proposition de Lozano & Garcia-Martinez (2010) qui utilise un Algorithme Evolutionnaire (AE) (Bäck, 1996a) pour perturber les solutions de départ dans une RLI. D'autres hybridations qui combinent plusieurs métaheuristiques sont présentes dans la littérature (Tseng & Liang, 2006). Ce travail est une approche hybride basée sur l'OCF, l'AG et la RL. L'OCF est appliquée pour produire une bonne population de départ pour l'AG. Par la suite, une alternance entre OCF et AG est exécutée et, à chaque itération, une RL est appliquée sur plusieurs solutions pour intensifier la recherche. Une autre approche qui combine plusieurs métaheuristiques est la proposition de Chen & Chien (2011). Dans ce travail, les auteurs proposent une nouvelle méthode qui combine l'AG, le RS, l'OCF et l'OEP. Premièrement, l'OCF génère différents groupes de fourmis. Ensuite, l'OCF et l'AG qui est hybridé avec le RS sont exécutés alternativement sur ces groupes séparément. Régulièrement, après un nombre défini d'alternances, l'OEP est exécutée sur tous les groupes pour permettre l'échange d'informations entre les différents groupes à travers la matrice de phéromone.

Un état de l'art regroupant un grand nombre de méthodes et de technique d'hybridation est présenté dans les travaux (Blum et al., 2008; Raidl et al., 2010; Bluma et al., 2011).

1.2 Les métaheuristiques parallèles

La communauté scientifique porte un intérêt croissant pour la programmation distribuée et massivement parallèle. En effet, ce type d'implémentation permet d'accroître non seulement l'efficacité des métaheuristiques, mais aussi la qualité des solutions trouvées.

1.2.1 La modélisation parallèle

La raison qui pousse les chercheurs à utiliser les modèles parallèles dans les métaheuristiques est l'importante demande en puissance de calcul pour les problèmes à résoudre. En effet, l'évaluation de la fonction objectif pour chaque solution est en général l'opération la plus coûteuse d'une métaheuristique (Luong, 2011). Nous pouvons distinguer trois modèles de parallélisation. La figure 1.1 décrit la combinaison de ces trois modèles (Talbi, 2009) :

- **Le modèle parallèle au niveau de l'algorithme** : il consiste à paralléliser des algorithmes (voir la partie 1 de la figure 1.1). Ce modèle ne dépend pas du problème traité. Si les algorithmes sont indépendants les uns des autres, leur parallélisation n'apporte qu'une accélération de leur exécution, Il n'y a pas d'amélioration de la qualité des solutions trouvées par rapport au modèle séquentiel. En revanche, si les algorithmes sont coopératifs (comme indiquer par les flèches à double sens de la figure 1.1, partie 1), leur parallélisation peut non seulement réduire leur temps d'exécution, mais aussi améliorer les solutions trouvées par rapport au modèle séquentiel.
- **Le modèle parallèle au niveau de l'itération** : ce modèle consiste à paralléliser la génération des voisins à chaque itération, indépendamment du problème traité (voir la partie 2 de la figure 1.1). Il améliore le temps d'exécution et non pas les solutions trouvées par rapport au modèle séquentiel. Son but est d'évaluer et de générer les solutions voisines parallèlement.

- **Le modèle parallèle au niveau de la solution** : ce modèle se concentre sur l'évaluation en parallèle d'une seule solution (voir la partie 3 de la figure 1.1). Il dépend du problème spécifique à gérer.

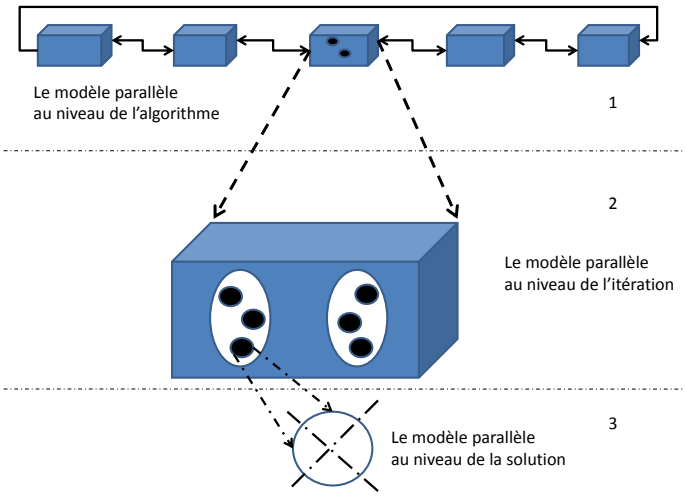


FIGURE 1.1 – Combinaison des trois modèles parallèles

1.2.2 Les architectures parallèles

Selon la classification de Flynn (Flynn, 1966), quatre architectures sont proposées pour gérer les modèles de parallélisation (voir Table 1.1) :

- **SISD** : c'est l'exécution d'une seule instruction sur une seule donnée. Cela représente l'architecture mono-processeur qui exécute les programmes séquentiellement. Cette architecture a tendance à disparaitre au profit de l'architecture multiprocesseur.
- **SIMD** : c'est l'exécution de la même instruction sur différentes données, cela représente l'architecture parallèle des données. Cette architecture est adoptée par les

cartes graphiques. Elle est très efficace dans l'exécution des algorithmes parallèles synchronisés qui contiennent un transfert régulier de données.

- **MISD** : c'est l'exécution de différentes instructions sur une seule donnée. Ce modèle représente la parallélisation des instructions. Cette architecture est adoptée par les pipelines.
- **MIMD** : c'est l'exécution de différentes instructions sur différentes données. C'est une architecture multiprocesseurs.

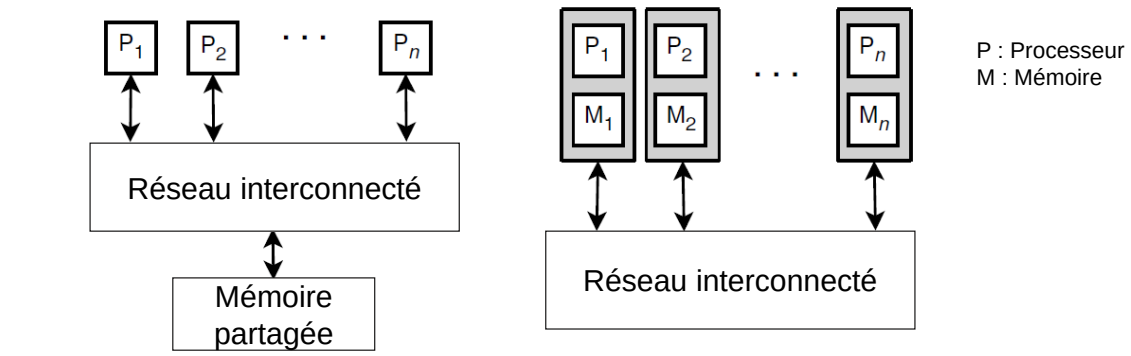
TABLE 1.1 – Classification de Flynn pour les architectures parallèles

	Single Data (une seule donnée)	Multiple Data (Plusieurs données)
Single Instruction (une seule instruction)	SISD	SIMD
Multiple Instruction (plusieurs instructions)	MISD	MIMD

Les architectures parallèles sont dominées par deux types :

- **Les architectures à mémoire distribuée** : chaque processeur gère sa propre mémoire (voir la partie (b) de la figure 1.2). Les différents processeur sont interconnectés entre eux par un réseau dont la topologie peut varier. Ce réseau est utilisé pour échanger les données entre les processeur.
- **Les architectures à mémoire partagée** : tous les processeur partagent la même mémoire (voir la partie (a) de la figure 1.2). Cette mémoire permet la communication et l'échange de données entre les processeur.

Actuellement, les architectures parallèles les plus dominantes sont les architectures hybrides qui combinent les architectures à mémoire partagée et celles à mémoire distribuée (voir la figure 1.3). Le principe est d'utiliser une architecture à base de mémoire partagée comme nœud principal d'une architecture à mémoire distribuée.



(a) Les architectures à mémoire partagée (b) Les architectures à mémoire distribuée

FIGURE 1.2 – Les architectures à mémoire distribuée et partagée (Talbi, 2009)

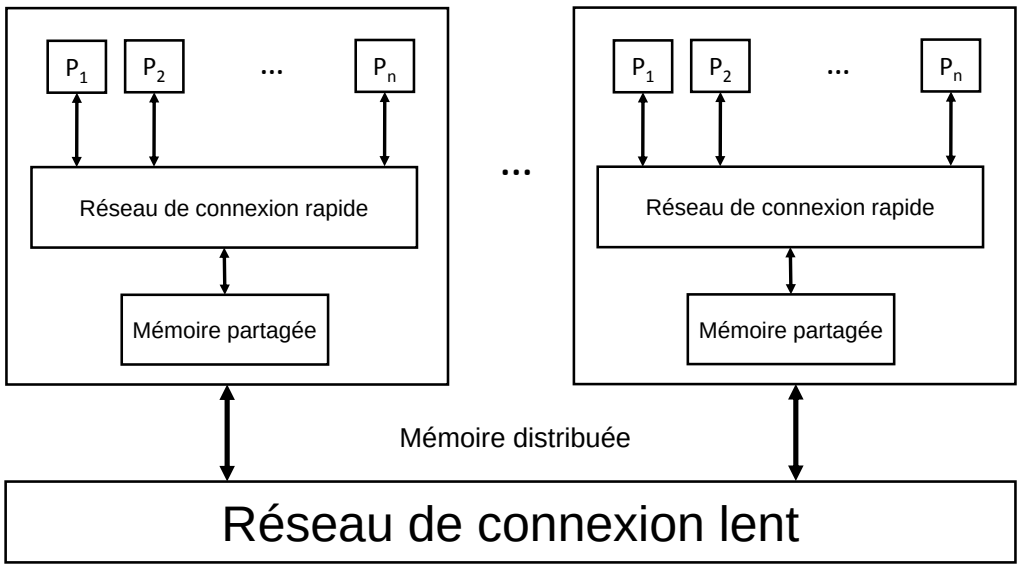


FIGURE 1.3 – Architecture hybride (Talbi, 2009)

1.2.2.1 Les métaheuristiques avec mémoire distribuée

L'architecture à mémoire distribuée utilisée dans notre travail est l'architecture NOW (Network Of Workstations). Cette architecture est potentiellement présente dans chaque université, laboratoire ou compagnie. Il suffit d'un ensemble de machines reliées entre elles pour créer cette architecture. Généralement, les machines disponibles dans ces structures sont sous-exploitées. Si on prend l'exemple d'une compagnie, les machines sont utilisées seulement pendant les heures de travail. En fonction des utilisateurs, les machines peuvent être sous-exploitées la majorité du temps. Le point faible de l'architecture NOW est la latence dans la communication entre les différentes machines. Cet inconvénient n'est pas un problème pour les métaheuristiques lorsque l'échange de données se fait d'une manière sporadique.

Dans ce travail, nous utilisons la bibliothèque de fonctions MPI (Message Passing Interface) pour assurer la parallélisation entre machines par passage de messages. MPI permet de gérer plusieurs communicateurs (processus autonomes). Chaque communicateur exécute son propre code et peut échanger des données via des sous-programmes de la bibliothèque MPI. Plus d'informations sur MPI peuvent être trouvées dans le livre de Snir et al. (1996).

1.2.2.2 Les métaheuristiques avec mémoire partagée

Actuellement, les architectures à mémoire partagée les plus connues sont le CPU (Central Processing Unit) et le GPU (Graphic Processing Unit). Dans ce travail, nous utilisons la programmation hybride hétérogène qui combine CPU et GPU. Depuis 2003, l'industrie des semi-conducteurs a pris deux trajectoires principales. La première trajectoire est celle des processeurs multi-cœurs (*the multicore trajectory*). Elle consiste à garder la rapidité d'exécution des programmes séquentiels tout en multipliant les coeurs du processeur. Cette trajectoire a donné naissance à des CPU comme le *Intel Core i7 microprocessor* composé de quatre coeurs performants pour la programmation séquentielle. La deuxième

trajectoire c'est celle des processeurs massivement multi-cœurs (*the many-core trajectory*). Elle consiste à augmenter le nombre de coeurs, mais avec des coeurs moins performants que ceux de la première trajectoire. Son objectif est d'augmenter le débit d'exécution des applications parallèles. Cette trajectoire a donné naissance à des GPU comme *NVIDIA Tesla M2050* qui atteint les 448 coeurs.

La plupart des ordinateurs personnels, qu'ils soient fixes ou portables, comportent un GPU. Ce processeur graphique a la particularité d'être un composant hautement parallèle avec une puissance de calcul très élevée. Ces propriétés ont été attribuées au GPU dans le but d'assurer des tâches telles que les rendus 3D et la gestion de la mémoire vidéo. Le potentiel de ce composant est sous exploité dans les applications non graphiques. Grâce à des langages tels que *CUDA* (Compute Unified Device Architecture) et *OpenCL* (Open Computing Language), les informaticiens sont capables d'implémenter des algorithmes parallèles en exploitant la structure et la puissance de calcul du GPU. Néanmoins, il faut une maîtrise de l'architecture GPU pour pouvoir implémenter des métaheuristiques qui exploitent au maximum les capacités du GPU. La figure 1.4, tirée du travail de Luong (2011), montre un modèle d'exécution simple d'un programme sur GPU.

Le CPU représente l'hôte et il est appelé *host*. Le GPU est considéré comme un co-processeur périphérique et il est appelé *device*. Tout d'abord, le CPU exécute ses instructions séquentiellement dans le *host*. Le *device* et le *host* n'ont pas accès à leur mémoire respective. Donc, nous devons allouer la mémoire que le *device* doit utiliser pour l'exécution des instructions sur le GPU. Une fois la mémoire allouée, il faut transférer les données du *host* vers le *device* pour l'exécution d'une manière parallèle. Cette exécution se fait à l'aide d'une portion de code nommée *kernel*. Ce *kernel* est invoqué par le *host* et exécuté dans le *device*. Finalement, le résultat est transféré du GPU vers le CPU.

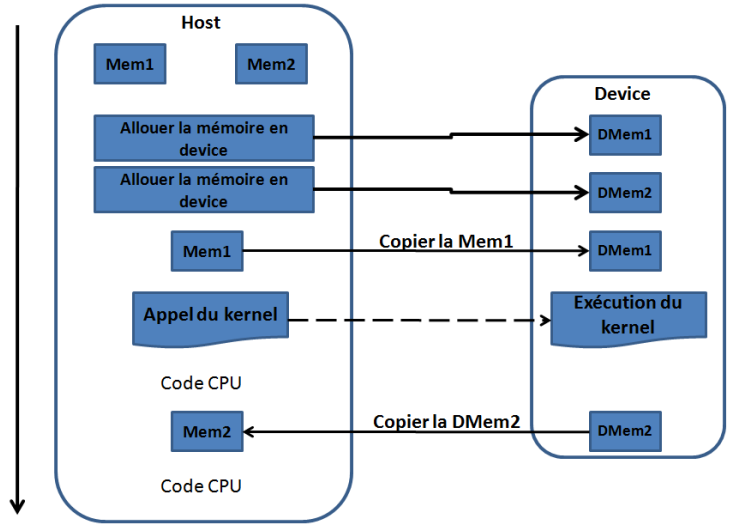


FIGURE 1.4 – Modèle d'exécution sur GPU (Luong, 2011)

Le CPU n'a pas beaucoup d'Unités Arithmétiques et Logiques (*UAL*). Par contre, ces unités sont très puissantes et elles utilisent une grande mémoire cache et une grande Unité de Contrôle (UC). Cela permet au CPU de stocker différentes informations dans le cache pour accélérer leurs accès, et l'unité de contrôle peut gérer différentes instructions complexes. Pour ces raisons, le CPU est très puissant et optimisé dans l'exécution séquentielle car il accède plus rapidement aux données.

Par contre, le GPU a beaucoup plus d'unités arithmétiques. Chaque groupe de ces unités a une mémoire cache limitée dans le but d'augmenter le débit de la mémoire. Les unités de contrôle sont simples. Le GPU a beaucoup plus de transistors dévoués au traitement des données et, en contrepartie, moins de transistors affectés au cache et aux unités de contrôle. Pour ces raisons, le GPU est très puissant dans l'exécution en parallèle. La figure 1.5 montre les différences entre les deux architectures.

L'exécution sur GPU se fait en système *SPMD* (*Singel Program Multiple Data*). Ce système signifie que le même code ou programme, contenu dans le *kernel*, s'exécute sur différentes données. Or, le GPU, a une architecture parallèle *SIMD* (*Singel Instruction*

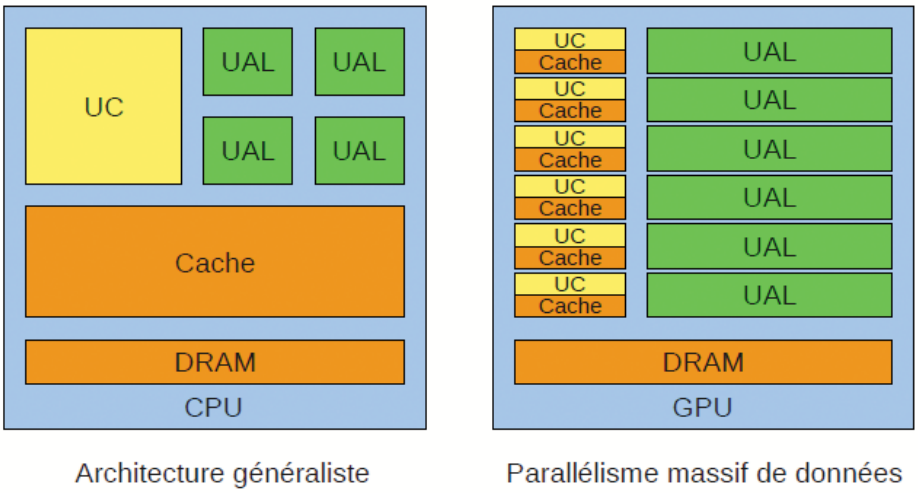


FIGURE 1.5 – Différence entre CPU et GPU

Multiple Data). Kirk & Hwu (2010) ont noté la différence entre *SIMD* et *SPMD*. Dans le système *SPMD*, le même programme n'est pas obligatoirement exécuté sur les mêmes parties des différentes données. Alors que dans le système *SIMD*, la même instruction est exécutée en même temps sur différentes données et par la même unité de contrôle. Par exemple, les instructions *If-Else* sont exécutées en *SPMD* sur différentes données. Par contre, en système *SIMD*, seul le *If* est exécuté sur différentes données puis la même chose pour le *Else* (Voir Figure 1.6). Brodtkorb et al ont montré l'importance du contrôle de cette divergence lors de la programmation sur *GPU* (André et al., 2013). La Figure 1.6 illustre le principe de la divergence dans le cas de l'utilisation d'un fragment de code *If-Else* :

La Figure 1.6 montre l'exécution de 32 *threads* au cours du temps. Nous pouvons remarquer que seuls les *threads* qui vérifient la condition *If* sont actifs. Les autres *threads* s'exécutent séquentiellement par rapport au premier groupe de *threads*, ce qui crée une divergence.

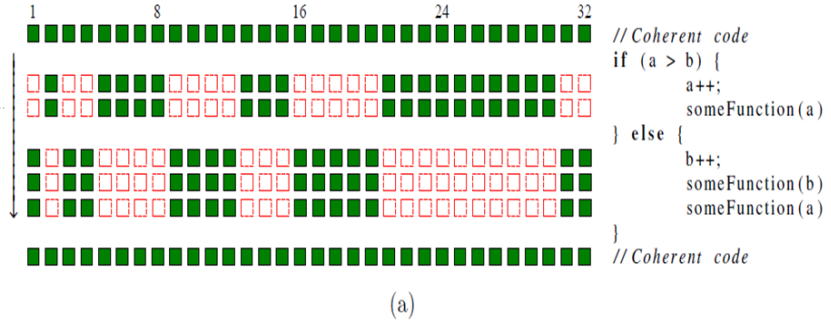


FIGURE 1.6 – Divergence d'execution (André et al., 2013)

1.2.3 Revue de la littérature des métaheuristiques hybrides parallèles

Les métaheuristiques hybrides distribuées (Idoumghar & Schott, 2009; Abdelkafi et al., 2016b) et massivement parallèles (Brévilliers et al., 2016) peuvent être classées aussi en deux catégories d'hybridation, à savoir l'hybridation à haut niveau et l'hybridation à bas niveau.

1.2.3.1 Les métaheuristiques hybrides distribuées

Les métaheuristiques distribuées ont la capacité d'améliorer la qualité des solutions proposées en plus de la réduction du temps d'exécution. La complexité des problèmes combinatoires et le temps de calcul qu'ils nécessitent les placent parmi les bons candidats pour la parallélisation.

La catégorie de l'hybridation à bas niveau est très peu présente dans la littérature. On peut trouver, par exemple, le travail de Cotta et al. (1995) qui proposent deux versions distribuées d'une hybridation d'un AG avec le Branch and Bound ($B\&B$) (Land & Doig, 1960). La version distribuée et hybridée à bas niveau propose de lancer une population d'individus et de remplacer le croisement dans l'AG par un $B\&B$. Chaque processus

exécute un $B\&B$ sur une partie différente de l'individu. Dans la même catégorie, on peut citer le travail de Talbi et al. (2001) qui proposent une parallélisation de l'OCF. L'idée est d'implémenter une mémoire centrale pour gérer toutes les communications des informations de la recherche dans le processus principal qu'on appelle communément *master process*. Dans ce travail, les informations de recherche sont composées de la matrice de phéromone et de la meilleure solution trouvée. À chaque génération, le master process diffuse la matrice de phéromone pour toutes les fourmis. Chaque processus représente une fourmi qui s'occupe de construire une solution faisable, puis applique une RT comme hybridation. Le processus envoie au master process la solution qu'il a trouvée et sa nouvelle matrice de phéromone locale. Ensuite, le master process met à jour les informations de recherche.

La catégorie de l'hybridation à haut niveau est beaucoup plus courante dans la littérature. La proposition de Kim (2002), d'un AG distribué, utilise plusieurs populations distribuées sur différent processus. Les informations sont centralisées dans le master process qui s'occupe des échanges des individus selon une stratégie de migration prédéfinie. Dans le même contexte, James et al. (2009a) ont introduit une RT parallèle coopérative. Cette approche initialise autant de solutions de départ qu'il y'a de processus disponibles. Chaque processus exécute une RT indépendamment des autres processus. Cette phase d'initialisation permet d'avoir de bonnes solutions de départ. Après cette phase, chaque processus exécute une génération qui consiste à lancer une RT. À la fin de chaque génération, le processus en cours compare sa solution avec le processus voisin. Si le processus voisin obtient de meilleurs résultats, le processus en court remplace sa propre solution par une mutation de la solution voisine. Cette méthode permet la coopération entre les processus pour effectuer une recherche plus efficace. En 2009, Idoumghar & Schott proposent deux algorithmes distribués pour résoudre le problème d'affectation de fréquences. Le premier algorithme utilise le modèle en îlots basé sur un AG hybride. Le deuxième algorithme utilise un modèle coopératif entre plusieurs RT. Une autre hybridation à haut

niveau, dans le contexte des métaheuristiques distribuées, est le travail de Subramanian et al. (2010) qui proposent une recherche à voisinage variable multi-start embarquée dans une RLI. Chaque processus traite une RLI en utilisant une recherche à voisinage variable avec différents paramètres. En 2015, Tosun propose un algorithme hybride et distribué. Cette proposition est composée de trois étapes. La première étape est de génèrer une bonne population de départ en utilisant un AG parallèle basé sur le modèle en ilôt. Chaque processus représente une île et, à chaque génération, le master process diffuse la meilleure solution trouvée vers toutes les îles. Ensuite, la deuxième étape est l'application d'une RT sur les individus de chaque île et la meilleure solution obtenue des deux premières étapes est utilisée comme une solution initiale pour une autre RT.

Plus d'informations sur les métaheuristiques hybrides distribuées peuvent être trouvées dans le travail de Hulianytskyi & Sirenko (2010).

1.2.3.2 Les métaheuristiques hybrides massivement parallèles sur GPU

Suite à l'apparition des langages de développement *CUDA* et *OpenCL*, différents chercheurs dans le domaine de la recherche opérationnelle se sont intéressés à la résolution des problèmes combinatoires sous GPU.

Dans la catégorie de l'hybridation à bas niveau, on peut citer le travail de Luong et al. (2010b). Ils ont proposé un AE hybridé avec une RL. La génération et l'évaluation des voisins sont parallélisées sous GPU. Chaque thread représente un voisin de la solution en cours. Leur contribution principale consiste à accélérer le transfert entre le host et le device avec une transformation appelée *two to one index* qui permet la compression de deux valeurs en une seule valeur, afin de diviser le temps de transfert par 2. Après le transfert, et grâce à une formule mathématique d'extraction, ils récupèrent les deux valeurs de départ pour effectuer les bons calculs. Encore dans cette même catégorie, une variante des algorithmes d'OCF appelée *cunning Ant System* (cAS)(Tsutsui, 2006), hybridée avec une RT, est développée par Tsutsui & Fujimoto (2011). La spécificité du cAS

est l'utilisation des solutions partielles, ce qui aide à éviter les stagnations prématurées. Ici l'OCF ne génère pas une nouvelle solution entière mais elle modifie plutôt la solution précédente. Cet algorithme utilise deux types de fourmis : la fourmi rusée (*cunning ant* nommé c-ant) et la fourmi donneuse (*donor ant* nommé d-ant). La c-ant construit sa solution en empruntant une partie de la solution existante appelée d-ant. Dans chaque itération, seule la meilleure solution est transférée entre le GPU et le CPU pour vérifier les conditions d'arrêts. Chaque thread représente une fourmi et applique une RT. La RT utilise 99,9% de la puissance de calcul. Une méthode appelée *Move-Cost Adjusted Thread Assignment (MATA)* est proposée pour réduire le temps d'exécution. Le calcul de chaque voisin dans la RT est exécuté dans un thread avec la méthode MATA. Dans ce calcul, quelques mouvements de voisinage sont calculés avec une complexité de $\theta(n)$ et d'autres mouvements avec une complexité de $\theta(1)$. Pour éviter la divergence, les auteurs ont proposé la séparation de ces mouvements en deux groupes qui s'exécutent en parallèle sous GPU. En 2012, deux algorithmes parallèles basés sur l'OCF sont proposés (Cáceres et al., 2012). La première version est une hybridation de l'algorithme d'OCF et la deuxième est un cAS hybride. Les deux algorithmes sont hybridés avec une RL qui est appliquée après chaque reconstruction d'une solution. Plusieurs expérimentations sont exécutées et la meilleure configuration est l'utilisation de 16 blocks indépendants de 32 threads où chaque thread représente une fourmi. Toujours dans l'optimisation par colonie de fourmis, Luong & Taillard (2012) ont publié un rapport de recherche qui propose une hybridation des fourmis rapide *Fast Ant (FANT)*. L'évaluation des voisins est exécutée en parallèle et des opérations de réduction dans la comparaison sont proposées. Cette réduction réduit la complexité de $\theta(n)$ à $\theta(log(n))$, avec n la taille des voisins évalués. Deux schémas parallèles sont conçus, l'évaluation d'un seul voisin et l'évaluation de plusieurs voisins.

Dans la catégorie de l'hybridation à haut niveau, Luong et al. (2010a) ont présenté dans leur article deux modèles pour les AE parallèles sur GPU. Les deux modèles associent un thread à un individu de la population. Le premier modèle est *Fermier-Travailleur*. Ce

modèle souffre des coûts de transfert entre CPU et GPU. Pour cette raison, un deuxième modèle a été proposé. Il s'agit du modèle en îles sur GPU qui considère chaque block comme une île. Dans la même catégorie, on peut trouver la proposition de Zhua et al. (2010). L'idée principale de ce travail est le développement d'une RT multi-start sur l'architecture SIMD du GPU. Cela signifie que plusieurs RT seront exécutées à partir de différentes solutions de départ. Une RT indépendante est affectée à chaque thread. Cette méthode permet de limiter la communication entre les threads (donc moins d'utilisation de mémoire), et de limiter les transferts CPU-GPU. 6144 threads (32 blocks de 192 threads par block) sont utilisés pour saturer le GPU. Une variable aléatoire est introduite dans chaque RT pour définir la technique de diversification à utiliser. Cette variable permet une meilleure exploration de l'espace de recherche. Après chaque itération, une des quatre opérations prédéfinies est sélectionnée aléatoirement pour générer la nouvelle solution courante. Ça peut être une mutation aléatoire de la solution courante, une mutation aléatoire de la meilleure solution trouvée, une nouvelle solution totalement aléatoire ou pas de changement sur la solution courante. Une autre hybridation à haut niveau est la RTI proposé par Delévacq et al. (2013). Ils ont publié un travail sur la *MMAS* (*The Max-Min Ant System*) qui est une variante de l'OCF. Deux approches ont été proposées dont l'une est une hybridation à haut niveau. Il s'agit de l'approche des multiples colonies de fourmis. Deux stratégies ont été proposées dans cette approche : *COLONY-bloc* et *COLONY-gpu*. Les meilleurs résultats de ce travail ont été atteint par la stratégie *COLONY-gpu*. En 2013, une RT multi-start est proposée par Czapinski (2013). Plusieurs RT sont exécutées en parallèle à partir de différentes solutions initiales et de différentes longueurs de liste taboue. Chaque RT est exécutée dans un block. L'évaluation et la génération des voisins sont parallélisées sur chaque thread. Avec ce design, chaque thread évalue et génère un voisin.

1.3 Objectif et motivation

L'objectif de ce travail de recherche est de proposer des métaheuristiques hybrides distribuées et massivement parallèles pour résoudre différents problèmes combinatoires. Une ou plusieurs approches seront proposées pour chaque problème. Le but est d'avoir une bibliothèque de métaheuristiques pour résoudre des problèmes combinatoires (trois types de problèmes dans le cadre de ce travail).

Nous avons pu constater, à travers les revues de la littérature effectuées, que la plupart des travaux qui utilisent le GPU sont dédiés à l'accélération. La motivation de ce travail est de combiner des approches massivement parallèles et distribuées dans une architecture hybride. Les méthodes proposées seront dédiées principalement à la qualité des solutions.

Conclusion

Dans ce chapitre, le contexte de notre travail a été présenté. Une idée générale sur les modèles pour rendre les métaheuristiques parallèles et les techniques d'hybridation de ces dernières a été expliquée. Un état de l'art sur les métaheuristiques hybrides, distribuées et massivement parallèles à été proposé. Finalement, les objectifs et les motivations de ce travail de doctorat ont été énumérés.

Dans le chapitre suivant, les niveaux de parallélisation que nous allons utiliser seront présentés, ainsi que leur mise en pratique, illustrée sur le problème du voyageur de commerce.

Chapitre 2

Concepts de base : cas d'étude du problème du voyageur de commerce

Dans le deuxième chapitre, nous présentons les cinq niveaux de parallélisation ainsi que les deux niveaux d'hybridation utilisés dans ce manuscrit. Ensuite, ces niveaux sont appliqués sur un problème académique qui est le problème du voyageur de commerce.

2.1 Les niveaux de parallélisation

Dans ce travail, cinq niveaux de parallélisation sont considérés. Comme présenté dans le chapitre 1, On trouve trois niveaux majeurs. Nous avons conservé la parallélisation au niveau des algorithmes et au niveau des solutions. Nous avons pris le parti de décomposer la parallélisation au niveau des itérations en deux niveaux. Le premier est la parallélisation des structures de voisinage pour les S-métaheuristiques. Le second est la parallélisation des tâches exécutées par les individus dans une P-métaheuristique. Cette décomposition a pour but de faire la distinction entre les méthodes utilisées pour les S-métaheuristiques et celles utilisées pour les P-métaheuristiques. Un autre niveau de parallélisation est éga-

lement utilisé. Il s'agit de la parallélisation des données proposée par Cecilia et al. (2013). Voici les cinq niveaux de parallélisation retenus pour ce travail.

2.1.1 Parallélisation des algorithmes

Le niveau de parallélisation des algorithmes consiste à paralléliser différentes métaheuristiques entre elles. Ce niveau peut concerner la parallélisation de différentes exécutions d'une S-métaheuristiques comme les RL multi-starts. Il peut également se réaliser avec la parallélisation de plusieurs colonies dans l'OCF ou encore la parallélisation par îlots dans l'AG. Ce niveau peut gérer aussi la parallélisation de différentes métaheuristiques hétérogènes.

La force de ce niveau de parallélisation est la coopération entre les différentes métaheuristiques parallèles. Cette coopération permet de changer le comportement de la métaheuristique et d'améliorer la qualité des solutions. Pour effectuer la coopération, quatre questions sont posées :

La première question est *comment* est-ce qu'on échange les données. Cette étape est conditionnée par la métaheuristique utilisée. Le protocole d'échange est la manière dont on va échanger les informations. Parmi les méthodes conventionnelles, on peut par exemple injecter une solution dans une population pour les P-métaheuristiques ou bien croiser une solution avec une autre pour les S-métaheuristiques.

La deuxième question est *quoi* échanger entre les processus. Cette question est dépendante du protocole d'échange choisi. Plusieurs informations peuvent être envoyées comme la meilleure solution, la solution courante, une solution aléatoire de la population ou encore une variante de la meilleure solution. L'information peut être définie, par exemple, avec l'âge de la solution dans la recherche ou sa rareté. D'autres informations peuvent être envoyées telles que la matrice de phéromone, la liste taboue, la longueur de la liste taboue, etc.

La troisième question est *quand* est-ce qu'on décide d'échanger les données. Cette décision est prise à l'avance et dépend du protocole d'échange. Elle permet la diversification ou l'intensification de la recherche.

La dernière question est *où* est-ce qu'on envoie l'information. La topologie d'échange peut être indépendante de toutes les autres questions. Plusieurs topologies sont connues telles que l'anneau, l'hyper-cube et la topologie complète.

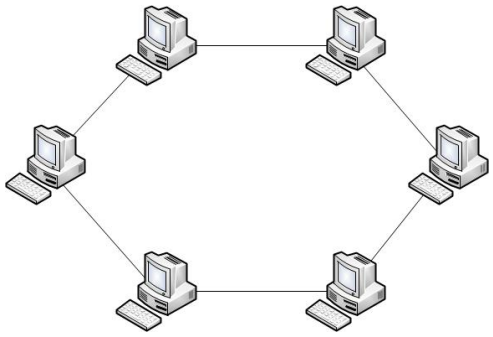


FIGURE 2.1 – Topologie en anneau

2.1.2 Parallélisation des structures de voisinage des S-métaheuristiques

La parallélisation des structures de voisinage est dédiée aux S-métaheuristiques. Ce niveau consiste à générer et évaluer les voisins d'une solution d'une manière parallèle au niveau de chaque itération.

La génération et l'évaluation des solutions voisines est une opération coûteuse pour les S-métaheuristiques. Cette opération peut être réalisée en parallèle. Le principe est que, à chaque itération, la solution est décomposée en différentes parties. La génération et l'évaluation des solutions voisines peuvent, par la suite, être appliquées à chacune de ces parties, en parallèle. De cette façon, les voisins sont générés et évalués d'une manière parallèle et indépendante. Cette parallélisation permet de gérer un ensemble de voisins de

très grande taille, ce qui est très fréquent dans les problèmes d'optimisation combinatoire du monde réel.

2.1.3 Parallélisation au niveau de la solution

Cette parallélisation est réalisée entre les éléments d'une même solution. Ce niveau peut intervenir, par exemple, dans la construction d'une solution d'une manière parallèle en la divisant en sous parties, ou encore dans la parallélisation du calcul effectué pour chaque partie de la solution, comme la mise à jour ou l'évaporation de la matrice de phéromones qui s'effectue pour une seule fourmi dans l'OCF.

2.1.4 Parallélisation des individus des P-métaheuristiques

La parallélisation des individus est dédiée aux P-métaheuristiques. Ce niveau gère les tâches indépendantes exécutées pour chaque individu dans une population, telles que le croisement pour les AG ou la reconstruction des solutions des fourmis pour l'OCF.

La parallélisation des structures de voisinage dans les P-métaheuristiques revient à paralléliser les tâches des individus de la population. Le principe est de décomposer la population en groupe d'individus qui effectuent des tâches en parallèle à chaque génération.

2.1.5 Parallélisation des données

Le niveau de parallélisation des données permet de traiter en parallèle des données indépendantes dans le problème ou les métaheuristiques. Ces données peuvent être relatives à la métaheuristique (matrice de phéromones, liste taboue, etc...) ou bien au problème (matrice de distances, matrice de flux, etc...).

2.2 Les niveaux d'hybridation

2.2.1 L'hybridation à bas niveau

Dans ce travail, l'hybridation à bas niveau est utilisée en ajoutant des heuristiques à l'intérieur des itérations d'une métaheuristique, ou dans l'échange de données entre les processus parallèles. Elle s'effectue en ajoutant une RL ou une RT dans les P-métaheuristiques. L'hybridation est également effectuée en remplaçant des opérateurs génétiques tels que la mutation par une RL.

2.2.2 L'hybridation à haut niveau

Dans cette thèse, l'hybridation à haut niveau est présente dans la parallélisation de plusieurs colonies, îlots ou RTI à travers le niveau de parallélisation des algorithmes. Il y'a une relation très étroite entre l'hybridation et la parallélisation. La parallélisation au niveau des algorithmes est généralement une hybridation à haut niveau.

2.3 Plateforme et architecture de tests

Tout au long des travaux présentés dans ce manuscrit, nous avons eu l'occasion d'utiliser un cluster de 12 machines connectées entre elles pour former une architecture NOW. La bibliothèque MPI est installée sur ces machines, ce qui permet d'effectuer des communications entre les machines. Pour la parallélisation sur GPU, nous utilisons le langage CUDA. Chaque machine du cluster dispose d'un GPU NVIDIA Geforce GTX680 et d'un CPU Intel Core processor i5-3330 CPU (3.00GHz) avec 4 GB de RAM. Les algorithmes proposés sont implémentés avec le langage C/C++ qui supporte MPI et CUDA.

2.4 Cas d'étude : problème du voyageur de commerce

Le premier problème traité dans ce travail est le problème du voyageur de commerce (Gutin & Punnen, 2002), ou *Traveling Salesman Problem* en anglais (TSP). C'est un problème académique facile à comprendre mais qui compte parmi les problèmes combinatoires les plus difficiles à résoudre. Il est utilisé pour résoudre de nombreux problème pratiques tels que des problèmes de tournées de véhicules (Laporte, 1992) et de cristallographie (Bland & Shallcross, 1989).

2.4.1 Présentation du problème du voyageur de commerce

Le TSP peut être défini avec un graphe complet (non orienté) $G = (V,E)$, où $V = \{1,\ldots,n\}$ est l'ensemble des sommets (villes), et $E = \{(i,j) : i,j \in V, i < j\}$ est l'ensemble des arêtes. d_{ij} est la distance Euclidienne entre les deux sommets i et j formant une arrête de E. Le problème consiste à trouver le trajet de longueur minimale passant par les n villes du problème et revenant au point de départ. A partir d'un ensemble de coordonnées, les distances d_{ij} entre les villes sont calculées. La solution est représentée par un vecteur d'entiers. Chaque entier représente une ville. Le vecteur commence avec la ville de départ et se termine par cette même ville. Le nombre de chemins possibles pour n villes est $\frac{(n-1)!}{2}$, donc le nombre de chemins possibles augmente de façon factorielle. Le TSP a été formulé par Desrochers & Laporte (1991) de la manière suivante :

$$Min \quad Z = \sum_{i<j} d_{ij} x_{ij} \tag{2.1}$$

$$s.c \quad \sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \quad (\forall k \in V) \tag{2.2}$$

$$U_i - U_j + (n-1)x_{ij} \leq n-2 \quad \forall (i,j \in \{2,\ldots,n\}, i \neq j) \tag{2.3}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E \tag{2.4}$$

$$U_i \in \mathbb{N} \quad \forall i \in V \tag{2.5}$$

L'équation 2.2 garantit que la solution sera un chemin qui passe exactement une fois par chaque ville de V. Les équation 2.3, 2.4 et 2.5 imposent à ce chemin de n'être formé que par un seul cycle couvrant toutes les villes, et non pas plusieurs cycles disjoints. Cela revient à former un cycle hamiltonien qui passe par toutes les villes de V.

2.4.2 Présentation de l'optimisation par colonie de fourmis

L'OCF, ou en anglais *Ant Colony Optimization* (ACO), est une méthode inspirée du comportement naturel des fourmis à la recherche de nourriture. Grâce à la similarité entre ce comportement et le problème du TSP, la première application d'OCF était proposée pour résoudre le problème de TSP (Dorigo, 1992; Dorigo et al., 1996) avec la variante de l'algorithme appelée *système de fourmis* ou en anglais *Ant System* (AS). Cette variante est divisée en trois étapes principales : la construction des tours, la mise à jour des phéromones et la mise à jours des probabilités. Ces trois étapes sont exécutées jusqu'à atteindre les conditions d'arrêts.

Intuitivement dans le comportement naturel, les fourmis recherchent la nourriture d'une manière aléatoire pour construire leur premier tour. Elles se déplacent d'un point à un autre jusqu'à ce qu'elles trouvent de la nourriture. Une fois la nourriture trouvée, les fourmis reviennent à leur point de départ. Cette étape correspond à l'initialisation. Dans le mécanisme de recherche, les fourmis déposent des phéromones tout au long du chemin qu'elles empreintent. Plus le chemin est utilisé par les fourmis, plus les phéromones s'accumulent. Pour utiliser ce concept dans le TSP, une matrice de phéromones est créée pour sauvegarder la quantité déposée pour chaque couple de villes. La quantité de phéromones entre i et j a une itération donnée t, nommée $\tau_{ij}(t)$, est calculée selon l'équation 2.6 :

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{N} \Delta\tau_{ij}^{k}(t) \quad \forall (i,j) \in E \tag{2.6}$$

avec $\Delta\tau_{ij}^k(t)$ la somme de phéromones déposées par la fourmi k sur le chemin entre les villes i et j. Cette valeur dépend de la longueur du tour C^k construit par la fourmi k ; $\Delta\tau_{ij}^k(t)$ est définie par l'équation 2.7 :

$$\Delta\tau_{ij}^k(t) = \frac{1}{C^k} \tag{2.7}$$

Une autre caractéristique de phéromones dans le comportement naturel est l'évaporation : les phéromones s'évaporent avec le temps si le chemin n'est pas choisi par les fourmis. Cette caractéristique est modélisée via le paramètre $0 < \rho \leq 1$ dans l'équation 2.8 :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t), \quad \forall (i,j) \in E \tag{2.8}$$

Dans le comportement naturel, les fourmis suivent la piste de phéromones la plus dense pour trouver le meilleur chemin vers la nourriture. Pour implémenter ce concept, une probabilité est définie dans l'équation 2.9 :

$$\Delta p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [n_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha [n_{il}]^\beta} \tag{2.9}$$

avec $n_{ij} = \frac{1}{c_{ij}}$ et c_{ij} la distance entre deux villes i et j, α et β sont des paramètres et N_i^k est l'ensemble des voisins réalisables. Un état de l'art complet sur l'OCF peut être trouvé dans le travail de Dorigo & Stützle (2004).

L'algorithme 1 présente la variante séquentielle de l'AS.

2.4.3 Conditions d'expérimentation

Le jeu de test utilisé est un ensemble d'instances connues de la bibliothèque TSPLIB (Reinelt, 1991) avec des tailles entre 51 et 200 villes. Tous les résultats sont présentés

Algorithme 1 AS séquentiel pour TSP

Entrée : N : nombre de fourmis ;
Sorties : $fourmi$: les fourmis de la population ; matrices des phéromones et de la probabilité
Initialisation de la population de fourmis ;
répéter
  pour i :=1 à N **faire**
    Construction de tour (pour la $fourmi_i$) ;
  fin pour
  pour i :=1 à N **faire**
    Evaporation des phéromones (pour la $fourmi_i$) selon 2.8 ;
  fin pour
  pour i :=1 à N **faire**
    Mise à jour des phéromones (pour la $fourmi_i$) selon 2.6 ;
  fin pour
  pour i :=1 à N **faire**
    Mise à jour de la probabilité (pour la $fourmi_i$) selon 2.9 ;
  fin pour
jusqu'à (condition d'arrêt)

en pourcentages de déviation par rapport à l'optimum. Cette déviation est calculée selon l'équation 2.10 :

$$Déviation = \frac{(f(s) - f(s^*)) \times 100}{f(s^*)} \tag{2.10}$$

avec $f(s)$ la valeur de la fonction objectif pour une solution s, et $f(s^*)$ celle pour la meilleures solution trouvée. L'optimum global de chaque instance est connu et présenté dans la bibliothèque TSPLIB.

2.4.4 Application des niveaux de parallélisation et d'hybridation sur le TSP

L'application des cinq niveaux de parallélisation est expérimentée sur le TSP avec l'algorithme d'OCF. Seul le niveau de parallélisation des algorithmes est exécuté sur l'architecture NOW et les 4 autres niveaux sont sur le GPU. Une hybridation à haut niveau et deux hybridations à bas niveau sont également expérimentées.

2.4.4.1 Le niveau de parallélisation des individus

Le niveau le plus facile à appliquer sur l'OCF et sur les P-métaheuristiques d'une manière générale est **le niveau de parallélisation des individus des P-métaheuristiques**. L'idée dans ce contexte est de paralléliser les tâches indépendantes effectuées par les fourmis.

Ce niveau est utilisé dans la construction des tours. Chaque fourmi est représentée par un thread et chaque thread construit une tournée parallèlement aux autres fourmis. Pour chaque thread, la fourmi choisit la prochaine ville à visiter parmi les villes qui n'ont pas été sélectionnées précédemment dans la tournée, selon une probabilité calculée par l'équation 2.9. L'algorithme 2 présente le kernel qui permet la construction des tournées pour chaque fourmi.

Algorithme 2 kernel de construction des tours

1: *Entrées* : c_{ij} : la matrice des distances ; *prob* : la matrice des probabilités ; T : la taille de l'instance ; v : l'indice de la ville d'arrêt et de départ
2: *Sorties* : P : la population de fourmis ; f : la fitness des fourmis
3: Obtenir l'indice du thread *idx* ; /*chaque *idx* représente une fourmi*/
4: $f[idx] := 0$;
5: Ajouter v à $P[idx]$ et le rendre tabou ;
6: $pos := v$;
7: **pour** j :=1 à (T-1) **faire**
8: Calculer la somme des probabilités S pour atteindre les villes restantes à partir de la ville d'indice *pos* ;
9: Générer une valeur aléatoire R entre 0 et S avec le générateur aléatoire de *CUDA* ;
10: $i := 0$; /* Compteur de villes */
11: $Pr := 0$; /* Compteur pour les probabilités */
12: **répéter**
13: Incrémenter i jusqu'à l'indice d'une ville non séléctionnée ;
14: Pr += $prob[pos][i]$; /* la probabilité d'emprunter le chemin entre *pos* et i */
15: **jusqu'à** ($Pr \geq R$)
16: Ajouter la ville i dans $P[idx]$ et la rendre tabou ;
17: $f[idx]$ += $c_{ij}[pos][i]$;
18: $pos := i$;
19: **fin pour**
20: Ajouter v dans $P[idx]$;
21: $f[idx]$ += $c_{ij}[pos][v]$;

2.4.4.2 Le niveau de parallélisation des données

Pour la mise à jour des phéromones (voir équation 2.6) et l'évaporation des phéromones (voir équation 2.8), l'utilisation du niveau de parallélisation des individus peut engendrer un problème d'accès concurrentiel. Ce problème arrive si plusieurs fourmis essayent de mettre à jour les phéromones d'un même couple de villes en même temps. Pour éviter ce problème, l'utilisation **du niveau de parallélisation des données** est possible. Ce niveau est proposé par Cecilia et al. (2013). Chaque couple de villes est géré par un thread. A chaque fois qu'une fourmi utilise le couple de villes du thread en cours, la quantité de phéromones est mise à jour. Ce niveau est utilisé également pour la mise à jour de la matrice de probabilités. La probabilité de chaque couple de villes est calculée dans un thread parallèle. Ce niveau est appliqué pour le kernel de mise à jour des phéromones (voir algorithme 3), d'évaporation des phéromones (voir algorithme 4) et de mise à jour des probabilités (voir algorithme 5).

Algorithme 3 Kernel de mise à jour des phéromones

1: *Entrées* : P : la population des fourmis ; f : la fitness des fourmis ; T : la taille de l'instance ; N : la taille de la population
2: *Sortie* : ph : la matrice de phéromones
3: Obtenir l'indice du thread idx ; /* chaque idx représente un couple de villes */
4: **pour** $i := 1$ à N **faire**
5: $d := f[i]$; /* d est la distance parcourue par la fourmi */
6: **pour** $j :=1$ à T **faire**
7: **si** l'indice dans la matrice ph de l'arc entre i et j est égal à idx **alors**
8: $ph[idx] := ph[idx] + (\frac{1}{d})$;
9: **fin si**
10: **fin pour**
11: **fin pour**

Algorithme 4 Kernel d'évaporation des phéromones

1: *Entrée* : ρ : coefficient d'évaporation

2: *Sortie* : ph : la matrice de phéromones

3: Obtenir l'indice du thread idx ; /* chaque idx représente un couple de villes */

4: $ph[idx] := (1\text{-}\rho) \times ph[idx]$; /* selon (2.8) */

Algorithme 5 Kernel de mise à jour des probabilités

Entrées : ph : la matrice de phéromones ; c_{ij} : la matrice des distances ; T : la taille de l'instance ; α et β sont des paramètres

Sortie : $prob$: la matrice des probabilités

Obtenir l'indice du thread idx ; /*chaque idx représente un couple de villes */

$total := 0$;

$arc := 0$;

si $cij[idx] \neq 0$ **alors**

$arc := (ph[idx])^{\alpha} \times (\frac{1}{c_{ij}[idx]})^{\beta}$;

fin si

$pos := \frac{idx}{T}$;

pour j :=0 à (pos-1) **faire**

$total \mathrel{+}= (ph[(pos \times T) + j])^{\alpha} \times (\frac{1}{c_{ij}[(pos \times T)+j]})^{\beta}$;

fin pour

pour $j := (pos + 1)$ à T **faire**

$total \mathrel{+}= (ph[(pos \times T) + j])^{\alpha} \times (\frac{1}{c_{ij}[(pos \times T)+j]})^{\beta}$;

fin pour

$prob[idx] := \frac{arc}{total}$;

2.4.4.3 Hybridation à bas niveau et parallélisation au niveau des structures de voisinage

Pour **l'hybridation à bas niveau** avec l'OCF, une RL parallèle ainsi qu'une nouvelle heuristique, nommée Fourmi Intelligente (FI), ont été utilisées. La RL parallèle est appliquée sur toutes les fourmis après la construction des tours. Dans cette RL parallèle, l'évaluation et la génération des voisins sont parallélisées sur GPU. Cette parallélisation consiste à représenter chaque ville par un thread. Chaque thread évalue toutes les permutations possibles avec les autres villes et sélectionne la meilleure permutation. Cela signifie que le thread évalue et génère le meilleur voisin de la ville en cours. La première moitié des fourmis exécute la permutation *switch* et la deuxième moitié exécute la permutation *2-opt*. La RL parallèle utilise **la parallélisation des structures de voisinage des S-métaheuristiques**.

La deuxième hybridation à bas niveau est l'heuristique des FI. Le but de cette heuristique est d'améliorer la fitness des fourmis en communiquant avec les autres fourmis. Cette heuristique exécute un nombre d'itération égal à la taille de l'instance sans considérer la ville de départ qui ne change pas. La figure 2.2 présente un exemple de l'heuristique FI pour une instance de 4 villes avec 3 itérations.

A chaque itération i, l'heuristique cherche la meilleure fourmi de la colonie. Par exemple, dans l'itération 2 de la figure 2.2, la meilleure fourmi est la troisième avec l'indice 2. Toutes les fourmis suivent le mouvement de $fourmi_2$ dans la position 2 indiquée par la flèche verticale dans la figure. La ville dans cette position de la $fourmi_2$ est la ville numéro 3. Par conséquent, $fourmi_0$ et $fourmi_1$ permutent leurs villes de façon à avoir la ville 3 dans la position 2. C'est pour cette raison que le nom de l'heuristique est FI car elles ont l'intelligence d'ajuster leurs tournées. Toutes les fourmis exécutent cette heuristique en parallèle avec **le niveau de parallélisation des individus**. Comme on peut le voir à travers cet exemple, après un certain nombre d'itérations, toutes les fourmis auront la même tournée. Pour éviter cette stagnation, les fourmis n'exécutent pas la

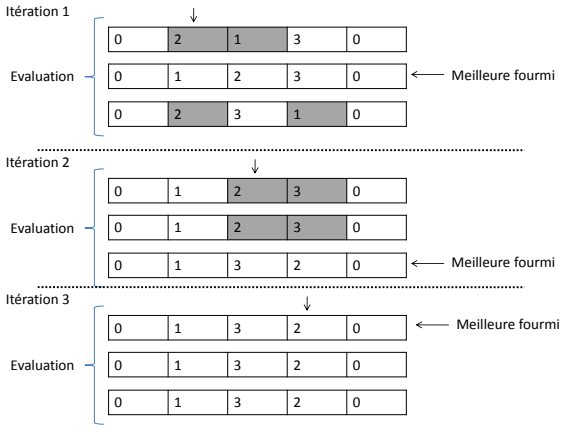


FIGURE 2.2 – Exemple illustrant le principe d'ajustement des tournées dans FI

TABLE 2.1 – Evaluation de l'heuristique FI

Instances	**Moyenne avec FI**	**Moyenne sans FI**
Eil51	**3,13%**	3,81%
Berlin52	**2,50%**	3,14%
Eil76	**5,64%**	6,35%
Pr76	**4,85%**	6,14%
KroA100	**4,67%**	5,26%

permutation lorsque les deux villes à permuter sont adjacentes (exemple dans figure 2.2 l'itération 1 pour $fourmi_0$). De plus, une mutation est effectuée à la fin de l'heuristique, i.e. une permutation est exécutée pour une fourmi aléatoire sur une position aléatoire pour diversifier la recherche. L'algorithme 6 présente le pseudo-code implémenté dans le kernel de l'heuristique des FI et exécuté pour tous les threads.

Le but de l'heuristique des FI est d'améliorer les résultats. Le tableau 2.1 montre la performance de cette heuristique pour une seule colonie. 25 tentatives sont exécutées pour chaque instance avec 100 itérations. Dans le tableau 2.1, les déviations moyennes des 25 tentatives sont présentées. Le tableau 2.1 montre l'efficacité de l'heuristique FI à travers 5 instances.

Algorithme 6 Kernel des FI

1: *Entrées* : A_{best} : la meilleure fourmi courante ; i : l'itération courante ; T : taille de l'instance
2: *Sorties* : A_{idx} : la fourmi du thread courant ; le coût de la fourmi A_{idx}
3: E := -1 ;
4: **pour** $j := i + 2$ à T **faire**
5: **si** $A_{idx}[j] == A_{best}[i]$ **alors**
6: $E := j$;
7: **fin si**
8: **fin pour**
9: **si** $E \neq$ -1 **alors**
10: Permuter de $A_{idx}[E]$ avec $A_{idx}[i]$;
11: Calculer le nouveau coût de la fourmi A_{idx} ;
12: **fin si**

2.4.4.4 Hybridation à haut niveau et parallélisation au niveau de l'algorithme

L'algorithme 7 présente toutes les étapes de l'algorithme implémenté. Cet algorithme correspond à la conception d'une seule colonie.

La dernière étape de l'approche est l'utilisation de MPI pour exécuter la méthode sur plusieurs machines. Cette étape est **une hybridation à haut niveau** en lançant plusieurs colonies de fourmis sur différents processus. Les colonies sont exécutées en parallèle en utilisant **la parallélisation au niveau des algorithmes**. Par exemple, si on exécute 3 processus, on va dupliquer notre algorithme 3 fois. Par conséquent, 3 colonies sont exécutées en parallèle. La bibliothèque MPI permet à l'algorithme proposé d'échanger des informations entre les différents processus afin d'améliorer les résultats.

Algorithme 7 Système de fourmis hybrides parallèles

1: *Entrée* : T : le nombre de villes de l'instance
2: *Sorties* : matrices des phéromones et des probabilités ; la population de fourmis
3: Initialisation des données et de la population de fourmis ;
4: Mise à jour des phéromones (**pour tous les arcs**) ;
5: Mise à jour des probabilités (**pour tous les arcs**) ;
6: **répéter**
7: Construction de la tournée (**pour toutes les fourmis**) ;
8: RL parallèle (**pour toutes les fourmis**) ;
9: **pour** i :=1 à (T-1) **faire**
10: FI (**pour toutes les fourmis**) ;
11: **fin pour**
12: Mutation (**pour toutes les fourmis**) ;
13: Evaporation des phéromones (**pour tous les arcs**) ;
14: Mise à jour des phéromones (**pour tous les arcs**) ;
15: Mise à jour de la probabilité (**pour tous les arcs**) ;
16: **jusqu'à** (critères d'arrêt satisfaits)

2.4.4.5 Parallélisation au niveau de la solution

Pour échanger les informations, l'algorithme choisit régulièrement la meilleure fourmi d'une colonie pour mettre à jour la matrice des phéromones d'un processus voisin selon une topologie en anneau.

Cette mise à jour des phéromones n'utilise pas le niveau de parallélisation de données car, dans une seule fourmi, chaque ville est visitée une seule fois. Dans ce cas, nous utilisons **le niveau de parallélisation des solutions** au niveau de chaque couple de villes de la fourmi. La mise à jour des phéromones est effectuée en parallèle pour chaque couple de villes qui se succèdent dans la solution. En effet, chaque couple de villes est unique car une ville est sélectionnée une seule fois dans la solution. Par conséquent, il n'y a pas de conflit d'accès à la mémoire puisque deux villes qui se succèdent n'apparaissent qu'une seule fois dans le chemin.

La figure 2.3 présente la conception finale de notre approche qui est publiée dans le travail de Abdelkafi et al. (2014).

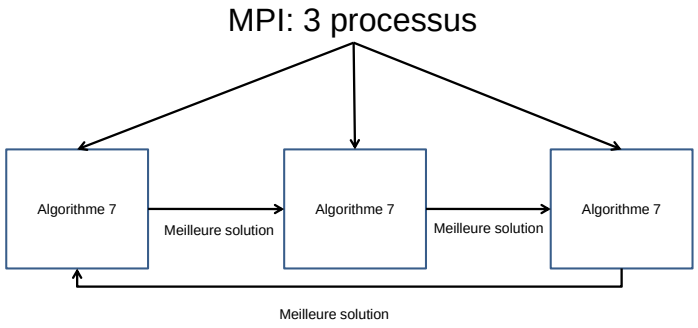


FIGURE 2.3 – Système de fourmis hybrides, parallèles en multi-niveaux

2.4.5 Expérimentation du TSP

Dans cette expérimentation, nous utilisons les 12 machines du cluster. 10 tests pour chaque instance sont effectués avec 10 instances de la bibliothèque TSPLIB. Le tableau 2.2 présente le meilleur résultat (MIN), le plus mauvais résultat (MAX), la moyenne des résultats (MOY) et le temps d'éxécution moyen des 10 tentatives (temps). Les paramètres utilisés sont $\alpha = 1$; $\beta = 2$; $\rho = 0,5$. 300 itérations sont exécutées pour chaque colonie, et chaque colonie contient 256 fourmis. 12 processus sont exécutés, soit un processus et une colonie par machines. Toutes les 10 itérations, le processus échange sa meilleure solution en utilisant la topologie en anneau. 60% des résultats proposés sont entre 0% et 3%. Pour les 10 instances, 9 résultats sont inférieurs à 5%.

TABLE 2.2 – Le système de fourmis hybrides, parallèles multi-niveaux

Instances	**MIN (%)**	**MAX (%)**	**MOY (%)**	**Temps (s)**
Eil51	0,99	3,02	1,98	10,57
Berlin52	0,03	2,33	1,07	14
St70	1,53	3,2	2,61	25,96
Eil76	2,83	5,04	4,21	28,02
Pr76	1,99	3,55	2,76	31,44
Rat99	3,53	7,8	6,19	35,7
KroA100	2,41	3,61	3,17	55,92
Bier127	1,25	2,58	1,87	87,72
Ch130	1,86	3,11	2,51	72,2
Ch150	2,84	3,75	3,42	76,8

Le tableau 2.3 propose une expérimentation sur 8 machines. Le premier objectif de cette expérimentation est d'évaluer l'impact de l'échange de données dans les mêmes conditions d'expérimentation. Le deuxième objectif est de calculer l'accélération entre une implémentation distribuée et une autre non distribuée. 4 instances sont expérimentées et la moyenne de 10 tentatives est présentée dans le tableau 2.3. MOY 1 est la colonne qui rassemble les valeurs de la version non distribuée et MOY 2 correspond à la version

TABLE 2.3 – Impact de la version distribuée sur les résultats

Instances	MOY 1(%)	MOY 2 (%)
Berlin52	2,72	**1,04**
Pr76	5,4	**3,08**
Bier127	2,53	**2,18**
Ch150	4,43	**3,91**

distribuée. Dans ces conditions, nous pouvons constater que l'échange de données est capable d'améliorer les résultats pour les 4 instances testées.

La dernière expérimentation est la comparaison de l'algorithme proposé avec d'autres méthodes de la littérature. Dans le tableau 2.4, quatre travaux de la littérature sont comparées avec notre proposition : un algorithme d'OCF pour résoudre le problème de TSP (Chirico, 2004) et d'autres méthodes pour résoudre le même problème (Cochrane & Beasley, 2003; Masutti & Castro, 2009; Somhom et al., 1997). Cinq instances de TSPLIB sont expérimentées. la colonne [*] correspond à notre approche et les résultats sont données en pourcentage de déviation par rapport à l'optimum global. Le test de Friedman (Friedman, 1937; Idoumghar et al., 2013) est utilisé sur les 5 instances avec un niveau de risque $\alpha = 5\%$. Le tableau 2.5 montre les résultats de ce test statistique pour chaque paire d'algorithmes avec une valeur critique de C=3,67. On peut déduire de l'analyse de ces valeurs que notre approche [*] est plus performante que les 4 algorithmes de la littérature expérimentés.

TABLE 2.4 – Comparaison avec la littérature

Instances	[*]	Chirico (2004)	Cochrane & Beasley (2003)	Masutti & Castro (2009)	Somhom et al. (1997)
Eil51	**1,98**	7,98	2,89	2,69	3,43
Berlin52	**1,07**	7,38	7,01	5,18	5,81
Eil76	4,21	12,08	4,35	**3,41**	5,46
Bier127	**1,87**	15,32	3	2,2	3,41
Ch130	**2,51**	24,15	2,82	2,82	2,82

TABLE 2.5 – Test de Friedman

Instances	Chirico (2004)	Cochrane & Beasley (2003)	Masutti & Castro (2009)	Somhom et al. (1997)
[*]	19	10	4	12
Chirico (2004)	-	9	15	7
Cochrane & Beasley (2003)	-	-	6	2
Masutti & Castro (2009)	-	-	-	8

2.4.6 Discussion

L'objectif premier de cette application sur le problème du TSP est de présenter et tester les bases des niveaux de parallélisation et d'hybridation qui seront utilisées dans ce travail de doctorat.

Pour le TSP, après plusieurs expérimentations, nous sommes partis du principe que nous ne pourrions plus améliorer les résultats avec cette approche. Néanmoins, les résultats obtenus sont, pour 9 instances sur 10, au-dessous des 5% en termes de déviation par rapport à l'optimum global.

Dans la suite du travail, munis des niveaux de parallélisation et d'hybridation définis dans ce chapitre, nous avons décidé d'entamer un autre problème académique qui est le problème d'affectation quadratique. L'objectif est d'appliquer ces concepts de base afin de produire des résultats compétitifs par rapport à la littérature.

Conclusion

Dans ce chapitre, les concepts de base de notre travail ont été présentés. Cinq niveaux de parallélisation et deux niveaux d'hybridation, utilisés tout au long de ce manuscrit, ont été détaillés. Une étude de cas à travers le TSP a été proposée pour illustrer les concepts de base présentés.

Dans le chapitre suivant, le problème d'affectation quadratique est présenté, ainsi que les différentes méthodes élaborées pour le résoudre.

Chapitre 3

Le problème d'affectation quadratique

Dans le troisième chapitre, nous présentons différentes métaheuristiques distribuées et massivement parallèles pour résoudre le problème d'affectation quadratique. D'abord nous décrivons les propositions distribuées avec une expérimentation pour valider leur efficacité. Ces propositions utilisent le niveau de parallélisation des algorithmes. Ensuite, nous ajoutons un autre niveau de parallélisation qui est celui des structures de voisinage. En combinant ces deux niveaux, nous allons tenter de résoudre des instances de très grandes tailles qui ne sont pas encore traitées dans la littérature.

3.1 Présentation du QAP

3.1.1 Description

Le problème d'affectation quadratique, ou en anglais *Quadratic Assignment Problem* (QAP), est un problème NP-difficile. Il représente un défi important pour différents domaines. Ce problème est connu pour ses multiples applications en chimie, transport, industrie et plusieurs autres disciplines. Nous pouvons trouver quelques travaux connus de l'application du QAP (Kadluczka & Wala, 1995; Bhaba et al., 1998; Polak, 2005; Zhang et al., 2005; Duman & Or, 2007; Ulutas & Konak, 2012; Wu & Hao, 2015). Le QAP

a été introduit par Koopmans & Beckmann (1957) pour modéliser l'affectation des ressources dans des localisations. L'objectif est de trouver le coût minimum d'affectation en considérant deux matrices de variables. La première matrice est celle des flux entre les ressources à affecter, et la deuxième matrice est celle des distances entre les localisations. Ce problème est formulé dans l'équation 3.1 :

$$\min_{p \in P} z(p) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)} \tag{3.1}$$

où f et d sont les matrices des flux et des distances, respectivement ; P est l'ensemble de toutes les permutations des n localisations possibles ; p_i est la localisation assignée à la ressource i. L'objectif est de minimiser $z(p)$, qui représente le coût total des affectations de la permutation p.

La figure 3.1 présente un exemple simple du problème du QAP. Dans cet exemple, les usines i et j représentent les ressources et les sites h et l représentent les localisations. F_{ij} est le flux entre les deux usines et D_{lh} est la distance entre les deux sites. L'objectif est de trouver l'affectation optimale des usines dans les sites de façon à minimiser le produit des flux multiplié par les distances.

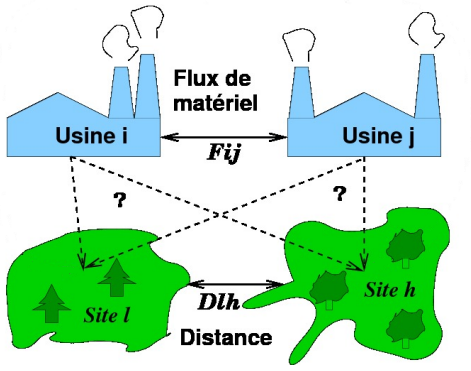


FIGURE 3.1 – Un exemple simple du QAP

3.1.2 Positionnement

Depuis l'introduction du QAP par Koopmans & Beckmann (1957), ce problème est devenu très important dans le domaine de l'optimisation combinatoire. Il peut être considéré comme un des problèmes les plus difficiles à résoudre à cause de sa complexité. Différentes métaheuristiques sont proposées pour résoudre le QAP (Misevicius, 2005; Stützle, 2006; Tseng & Liang, 2006; Puris et al., 2010; Hussin & Stützle, 2014). Un état de l'art sur le QAP peut être trouvé dans l'article de Loiola et al. (2007).

L'une des méthodes les plus efficaces pour résoudre le QAP est la recherche taboue robuste, nommée Ro-TS (de l'anglais *Robust Tabu Search*) (Taillard, 1991). Cette métaheuristique propose une réduction significative de la complexité d'évaluation en utilisant la matrice *delta* (Taillard, 1991; James et al., 2009b; Benlic & Hao, 2013). Plusieurs travaux sont basés sur la Ro-TS pour résoudre le QAP (Benlic & Hao, 2013; James et al., 2009a,b).

James et al. (2009b) proposent cinq variantes de la RT étendue à partir de la Ro-TS et séparées en deux catégories. La première catégorie contient des variantes qui perturbent la RT en modifiant quelques paramètres de la recherche. Cette catégorie a la particularité de continuer l'exploration avec la solution courante. La meilleure variante de cette catégorie est appelée *tabu tenure modification tabu search*. Pour la seconde catégorie, une nouvelle solution de départ remplace la solution en cours. Deux variantes de cette catégorie se distinguent, nommées *best solution found tabu search*, et *diversification tabu search*. La meilleure variante est la *diversification tabu search* et toutes les variantes sont plus efficaces que la Ro-TS. En 2013, Benlic & Hao proposent une Recherche Locale Itérative (RLI) avec une stratégie qui lui permet de s'échapper des optimums locaux en se basant sur l'historique de la recherche. Selon l'évolution de la recherche, la RLI sélectionne un des différents degrés de perturbation pour réussir à s'échapper de l'optimum local. En 2014, deux versions de RT et deux versions de RS sont comparées dans le travail de Hussin & Stützle. Les auteurs observent que la taille de l'instance est primordiale pour sélec-

tionner le bon algorithme parmi les quatres algorithmes proposés. Dans la même année, Tosun (2014) propose un nouvel opérateur de croisement testé sur plusieurs instances de la bibliothèque QAPLIB (Burkard et al., 1997). En 2015, Benlic & Hao proposent un algorithme mémétique appelé BMA (de l'anglais *Breakout Memetic Algorithm*) pour résoudre le QAP. Les auteurs combinent un ancien algorithme qui est la RLI de Benlic & Hao (2013) avec un croisement uniforme.

On peut observer à travers la littérature que la plupart des métaheuristiques efficaces qui sont dédiées à la résolution du QAP sont basées sur les RT.

La complexité du QAP et la difficulté de son espace de recherche donnent une grande importance à la modélisation parallèle des métaheuristiques dans sa résolution.

Les métaheuristiques distribuées sont sous-exploitées dans la résolution du QAP. Très peu de travaux proposent des solutions distribuées telles que le travail de Taillard (1991) qui propose une parallélisation de l'évaluation des voisins entre les différents processus, ou encore le travail de James et al. (2009a) qui proposent d'exécuter plusieurs RT sur différents processus avec un échange de données pour la diversification.

Du coté des propositions massivement parallèles pour résoudre le QAP, Tsutsui & Fujimoto (2009) proposent un AG parallèle qui utilise le langage CUDA. Deux contributions principales sont développées pour accélérer l'algorithme. La première contribution est la gestion des mémoires du GPU en maximisant l'utilisation des mémoires partagées. L'utilisation de cette mémoire permet de mieux accélérer grâce à son temps d'accès très réduit par rapport à celui de la mémoire globale. La deuxième contribution est la génération d'un nombre maximum de threads. Les auteurs représentent un individu dans un tableau d'entiers non signés, ce qui consomme moins de mémoire et donc permet de lancer plus de threads qu'avec des individus représentés par des entiers signés. Les auteurs comparent les performances de l'algorithme obtenues avec un CPU Intel Core i7 965 par rapport à celles obtenues avec un GPU NVIDIA GeForce GTX285. Huit instances du QAPLIB sont expérimentées avec des instances de taille entre 20 et 40. Un AG séquentiel est im-

plémenté pour l'exécution CPU. Ils ont obtenu un facteur d'accélération qui va de ×2,9 à ×12,6 entre CPU et GPU ($\frac{GPU}{CPU}$). La moins bonne accélération est obtenue avec *tai35b* (instance de taille 35) pour un temps de 2,51 secondes sur GPU. La meilleure accélération est obtenue avec l'instance *Kra30b* (de taille 30) pour un temps de 1,332 secondes sur GPU. En 2013, Luong et al. proposent une modélisation parallèle d'une RL sur GPU appliquée sur différents problèmes dont le QAP. Dans ce travail, la génération des voisins est effectuée directement sur GPU pour optimiser les transferts de données entre le CPU et le GPU. Les matrices des flux et des distances sont stockées dans la mémoire de texture sur le GPU afin d'accélérer leur accès. Une comparaison est effectuée sur quatre couples de CPU-GPU (*core 2 duo T5800* contre *GeForce 8600M GT 32 GPU cores* ; *core 2 Quad Q6600* contre *GeForce GTX 128 GPU cores* ; *Xeon E5450* contre *GeForce GTX280 avec 240 GPU cores* et finalement *Xeon E5620* contre *Tesla M2050 448 GPU cores*). Des instances entre 30 et 100 du QAPLIB sont utilisées dans l'expérimentation. Pour le GPU *GTX280*, l'utilisation de la mémoire de texture permet une meilleure accélération avec *tai100a* (de taille 100) (×10,9 en utilisant le GPU sans la mémoire de texture dans un temps de 5,5 secondes, et ×16,5 en utilisant le GPU avec la mémoire de texture dans un temps de 3,7 secondes). En 2014, Chaparala et al. proposent une RLI déployée avec l'architecture SIMD du GPU. Chaque thread gère une solution et utilise la structure de voisinage 2-opt. L'expérimentation est exécutée sur un cluster dans lequel chaque machine est équipée de deux *Intel Xeon 8-Core 64-bit E5 processors* (2,7 GHz) et un *Intel Xeon Phi Co-processor* (1,1 GHz). Le cluster a 128 noeuds et chaque noeud est équipé d'un GPU *NVIDIA K20*. Le nombre de threads est de 6144 répartis sur 24 blocs. 17 instances sélectionnées du QAPLIB sont expérimentées avec des tailles entre 30 et 100. La déviation par rapport à l'optimum global varie entre 0% (*tai30b* (de taille 30), *tai64c* (de taille 64)) et 2,5% pour *tai60a* (de taille 60). La proposition est comparée à deux algorithmes (un algorithme distribué et une RT sur GPU proposée par Zhua et al. (2010)). La proposition des auteurs obtient une accélération de ×16,31 par rapport à l'algorithme distribué et de

×58,61 par rapport à la RT de Zhua et al. (2010). Un état de l'art sur les propositions massivement parallèles pour résoudre le QAP peut être trouvé dans l'article de Abdelkafi et al. (2016a).

Dans ces travaux, les auteurs ont pour seul objectif de maximiser le facteur d'accélération, sans considérer la qualité des solutions obtenues. Plus généralement, nous pouvons remarquer que la plupart des travaux sur le GPU ne cherchent pas à utiliser les possibilités offertes par la parallélisation pour améliorer, en plus du temps d'exécution, la qualité des solutions trouvées.

3.2 Proposition des métaheuristiques distribuées

En se basant sur l'état de l'art réalisé pour le QAP dans la section précédente, nous avons décidé de proposer un algorithme distribué basé sur une RTI, en ayant exclusivement pour objectif l'amélioration des résultats grâce à l'environnement distribué. Pour ce faire, nous avons utilisé une hybridation à haut niveau, ainsi que le niveau de parallélisation des algorithmes.

Quatre variantes ont finalement été proposées. Ces variantes sont décrites dans les sections qui suivent.

3.2.1 Recherche taboue itérative hybride distribuée

La recherche taboue itérative hybride distribuée, que nous avons nommé D-HITS (de l'anglais *Distributed Hybrid Iterative Tabu Search*), est la première variante décrite dans ce chapitre. L'objectif est d'exécuter différentes recherches taboues iteratives hybrides, que nous avons nommé HITS (de l'anglais Hybrid Iterative Tabu Search), distribuées sur différents processus parallèles. Dans cette version, il n'y a pas d'échange d'informations entre les processus. Chaque HITS s'exécute indépendamment des autres avec différentes solutions de départ, et différents paramètres tels que des valeurs de seuil (*L1, L2* dans

l'algorithme 8). Le but de cette variante est d'avoir une version de base qui nous permettra d'étudier l'impact de l'échange de données entre les processus.

La RT représente le mécanisme d'intensification de notre proposition. Elle produit la meilleure solution possible après un ensemble de mouvements au sein de l'espace de recherche. Après chaque RT, une diversification adaptative est appliquée pour la meilleure solution trouvée. Le but de cette diversification est de découvrir une nouvelle région prometteuse de l'espace de recherche pour l'exploration de la RT suivante. Cette étape ajoute **le niveau d'hybridation à bas niveau**.

Pour chaque processus, l'historique de recherche est utilisé pour appliquer des mesures préventives afin d'eviter la stagnation. Un compteur w est initialisé à 0 et, après chaque RT sans amélioration, le compteur w est incrémenté de 1. Si une amélioration est enregistrée, w est remis à zéro. La solution est perturbée après un ensemble de RT sans amélioration. De cette façon, si l'algorithme est piégé dans une région de l'espace, alors, en plus de la diversification appliquée après chaque RT, une partie de la solution est perturbée pour débloquer la recherche. Si w continue à augmenter, une relocalisation complète est nécessaire pour explorer d'autres régions de l'espace de recherche. L'algorithme 8 présente le pseudo-code de D-HITS et l'algorithme 9 donne la RT utilisée dans notre proposition qui est tirée du travail de Taillard (1991).

Comme montré dans l'algorithme 8, en ligne 20, dans chaque itération globale de la D-HITS (itération entre deux RT consécutives), une diversification est appliquée sur la meilleure solution trouvée. Pour un ensemble d'itérations globales successives, la meilleure solution trouvée peut rester la même. Par conséquent, si la diversification utilisée est déterministe, alors elle produira la même nouvelle solution de départ à chaque exécution. Pour cette raison, la diversification doit être capable de produire des résultats différents à partir d'une même solution. Cette spécificité est, notamment, celle de la Diversification proposée par Glover (1998) (DG). La procédure de diversification traite une solution (dans notre cas, elle traite la meilleure solution trouvée à chaque itération globale) et exécute

Algorithme 8 D-HITS pour chaque processus

Entrées : *perturb* : % de la perturbation ; *L1, L2* : les seuils des mesures de prévention
Sorties : *cost* : le coût de la solution courante ; *Fcost* : le meilleur coût trouvé ; *sc* : la solution courante ; S_{best} : la meilleure solution trouvée
Initialisation de la solution du processus courant ;
$w := 0$; /* Le compteur qui définit l'état de l'historique de recherche */
Stagnation := **false** ;
répéter
    RT /* voir algorithme 9 */
    si *cost* < *Fcost* **alors**
        /* amélioration */
        Stagnation := **false** ; $w := 0$; *Fcost* := *cost* ;
        Mise à jour de S_{best} avec *sc* ;
    sinon
        w++ ;
    fin si
    /* Condition de stagnation */
    si w == *L2* **alors**
        Stagnation := **true** ;
    fin si
    si *Stagnation* == **false alors**
        /* pas de stagnation */
        Mise à jour de *sc* avec la **Diversification** de S_{best} ; /* voir algorithme 10 */
    sinon
        /* Stagnation */
        Re-localisation de la *sc* ;
        Stagnation = **false** ;
        $w := 0$;
    fin si
    si w == *L1* **alors**
        Perturbation de *sc* avec le paramètre *perturb* ;
    fin si
jusqu'à (Condition d'arrêt)

Algorithme 9 RT

1: **Entrées** : *cost* : le coût de la solution courante ; *iterRT* : le nombre d'itérations que la RT exécute
2: **Sorties** : *sc* : la solution courante ; *Tcost* : le meilleur coût trouvé dans la RT ; *Tsolution* : la meilleure solution trouvée dans la RT
3: **pour** $i := 0$ à *iterRT* **faire**
4: **si** les mouvements sont tabous mais satisfont tous les critères d'aspiration, ou s'ils ne sont pas tabous et trouvent un meilleur *Tcost* **alors**
5: Enregistrer les indices des meilleures permutations qui satisfont toutes les conditions ;
6: **fin si**
7: Mise à jour de la liste taboue ;
8: Mise à jour de *sc* ;
9: **si** le *cost* est meilleur que le *Tcost* **alors**
10: Mise à jour de *Tsolution* avec *sc* ;
11: **fin si**
12: Mise à jour des coûts de la matrice delta ;
13: **fin pour**

un ensemble de permutations en suivant la valeur de *step*. Cette valeur change d'une itération globale à une autre. De cette façon, même si la meilleure solution trouvée reste statique, la nouvelle solution de départ produite sera différente. L'algorithme 10 présente le pseudo-code de la DG.

Le mécanisme de prévention est basé sur l'historique de la recherche. C'est une manière intelligente d'explorer l'espace de recherche. L'idée principale est de trouver des régions prometteuses de l'espace pour chaque processus dans le cas d'une stagnation. Après un ensemble d'itération globale sans amélioration, w atteint son premier seuil *L1* (algorithme 8 ligne 27 à 29). La première mesure pour débloquer la recherche du processus est l'application d'une perturbation aléatoire sur une portion de la solution obtenue après la DG. Cette mesure suppose qu'il n'y a plus d'amélioration possible avec l'évolution actuelle de la recherche. Par conséquent, la première mesure de prévention est appliquée. Si cette mesure permet une amélioration de la meilleure solution trouvée, alors le D-HITS continue sa recherche et le compteur w est remis à zéro, sinon, le compteur continue à être incrémenté.

Algorithme 10 La stratégie de DG

```
1: Entrées : S_best : la meilleure solution trouvée ; step : la valeur qui détermine la permutation ; T : la taille de l'instance
2: Sortie : sc : la solution courante
3: pos := 0 ;
4: pour i := step à 1 faire /* décrémentation de i par 1 */
5:     pour j := (i-1) à T faire /* incrémentation de j avec step */
6:         sc[pos] := S_best[j] ;
7:         pos++ ;
8:     fin pour
9: fin pour
10: si step == T - 1 alors
11:     Réinitialisation de step
12: sinon
13:     step++ ;
14: fin si
```

Lorsque w atteint le deuxième seuil *L2* (algorithme 8 ligne 14 à 17), l'algorithme suppose qu'une relocalisation totale de la solution courante est nécessaire car la recherche n'arrive plus à trouver de meilleure solution. Dans ce cas, l'algorithme ignore la DG et génère une nouvelle solution aléatoire (algorithme 8 ligne 22 à 26). Le but de cette deuxième mesure préventive est d'explorer la possibilité qu'une meilleure solution peut exister dans une autre région de l'espace de recherche.

3.2.2 D-HITS avec la coopération à travers les distances

Nous proposons deux versions de D-HITS avec coopération de distances que nous avons nommées DISCO-HITS (de l'anglais *DIStance COoperation Hybrid Iterative Tabu Search*)(Abdelkafi et al., 2015a) : La première version, nommée *DISCO-HITS-DG*, utilise la DG et la deuxième version, nommée *DISCO-HITS-UX*, utilise le croisement uniforme connu sous l'abréviation de UX. Différents HITS sont exécutés dans un environnement distribué et à partir de différentes solutions de départ. Un échange de données entre les processus est effectué en suivant une topologie en anneaux.

L'échange d'informations classique consiste généralement à envoyer la meilleure solution du processus courant vers un processus voisin. Cet échange permet au processus voisin d'améliorer sa recherche si l'information reçue est capable d'améliorer sa propre meilleure solution.

Nous proposons d'introduire un nouveau mécanisme d'échange d'informations pour le QAP. Le processus envoie sa solution courante et reçoit la solution courante du processus voisin. L'idée est de calculer la similarité entre les deux solutions à chaque position pour définir une distance (algorithme 11 ligne 11-15). Selon cette distance, chaque processus prend une décision et suit une série d'instructions pour continuer la recherche (diversification, perturbation ou relocalisation). Chaque processus exécute une HITS et l'évolution de chaque processus dépend de l'évolution de son voisin. L'objectif est d'explorer intelligemment différentes régions de l'espace de recherche.

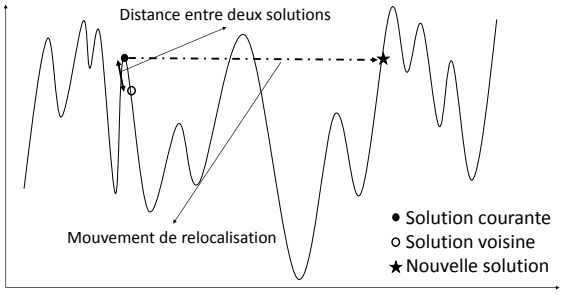


FIGURE 3.2 – Mouvement de relocalisation en utilisant la distance

La figure 3.2 illustre le mouvement de relocalisation effectué qui utilise le concept de distance. Si les deux solutions (courante et voisine) sont très proches dans l'espace de recherche, la variable *level* prend la valeur 2 (algorithme 11 ligne 22). Cette valeur indique à l'algorithme que la solution courante doit exécuter une relocalisation pour découvrir une nouvelle région de l'espace de recherche (algorithme 11 ligne 31).

L'algorithme 11 est dupliqué pour tous les processus. Il exécute une succession de RT (algorithme 9). Après chaque RT, l'algorithme 11 enregistre la nouvelle meilleure

Algorithme 11 Algorithme du DISCO-HITS

Entrées : *perturb* : % de perturbation ; T : taille de la solution ; S_{EX} : la solution echangée
Sorties : *cost* : coût de la solution courante ; *Fcost* : le meilleur coût trouvé ; *sc* : la solution courante ; S_{best} : la meilleure solution trouvée
Initialisation de la solution du processus courant ;
répéter
    RT /* voir Algorithme 9 */
    si *cost* < *Fcost* **alors**
        Fcost := *cost* ;
        Mise à jour de S_{best} avec *sc* ;
    fin si
    level := 0 ; *counter* := 0 ;
    Echange de *sc* entre les processus (topologie en anneaux) ;
    pour i := 0 à T /* calcul des distances */ **faire**
        si *sc[i]* == S_{EX}*[i]* **alors**
            compteur++ ;
        fin si
    fin pour
    si *compteur* < $\frac{T}{4}$ **alors**
        level := 0 ; /* grande distance entre les processus */
    sinon
        si *compteur* < $\frac{3\times T}{4}$ **alors**
            level := 1 ; /* distance moyenne entre les processus */
        sinon
            level := 2 ; /* petite distance entre les processus */
        fin si
    fin si
    si *level* == 0 **alors**
        Mise à jour de *sc* avec la **diversification** de S_{best} (algorithme 10 ou algorithme 12) ;
    sinon
        si *level* == 1 **alors**
            Perturbation de *sc* avec le paramètre *perturb* ;
        sinon
            Relocalisation de *sc* ;
        fin si
    fin si
jusqu'à (condition d'arrêt)

solution dans le cas d'une amélioration. L'étape suivante de l'algorithme est l'envoi de la solution courante et la réception de cette même information du processus voisin. C'est grâce à cette information que l'algorithme va pouvoir déterminer la distance entre les deux solutions voisines. Selon cette distance, l'algorithme prend la décision adéquate. Cette décision peut être : l'exécution d'une diversification (algorithme 10 pour DISCO-HITS-DG ou bien algorithme 12 pour DISCO-HITS-UX), la perturbation de la solution (ligne 29, algorithme 11) ou finalement une relocalisation de la solution (ligne 31, algorithme 11).

Pour le DISCO-HITS-DG, notre approche applique la DG (algorithme 10) et pour le DISCO-HITS-UX, la diversification appliquée est la UX présentée dans l'algorithme 12.

Cette diversification traite la solution voisine et s'exécute en suivant une séquence représentée par un vecteur de sélection. Ce vecteur est perturbé avant chaque application de l'UX. De cette manière, même si la meilleure solution reste statique, la nouvelle solution générée est différente à chaque fois. L'exemple suivant est une application simple de l'UX (algorithme 12) :

Initialisation :

— $select$ = (0,1,1,0,1,0).

— $index$ = (0,0,0,0,0,0).

— $S_{current}$ = (1,2,0,3,5,4).

— S_{best} = (3,5,1,0,4,2).

Première boucle :

— $S_{current}$ = (1,-1,-1,3,-1,4).

— $index$ = (0,1,0,1,1,0).

Deuxième boucle :

— $S_{current}$ = (1,5,-1,3,-1,4).

— $index$ = (0,1,0,1,1,1).

Troisième boucle :

— $S_{current}$ = (1,5,0,3,2,4).

— $index$ = (1,1,1,1,1,1).

Algorithme 12 Croisement uniforme UX

Entrées : S_{Ex} : la meilleure solution trouvée par le processus pour l'échange ; T : la taille du problème ; *select* : séquence qui définit les points de croisement avec un ratio de 0.5 ; *index* : le filtre qui permet de rendre la solution faisable, initialisé à 0
Sortie : *sc* : la solution courante
Perturbation de *select* ;
/* première boucle */
pour $i := 0$ à T **faire**
    si $select[i] == 0$ **alors**
        sc[i] := sc[i] ;
        $index[sc[i]] := 1$;
    sinon
        sc[i] := -1 ;
    fin si
fin pour
/* deuxième boucle */
pour $i := 0$ à T **faire**
    si $select[i] == 1$ **et** $index[S_{Ex}[i]] == 0$ **alors**
        sc[i] := $S_{Ex}[i]$;
        $index[sc[i]] := 1$;
    fin si
fin pour
/* troisième boucle */
pour $i := 0$ à T **faire**
    si sc[i] == -1 **alors**
        pour $k := 0$ à T **faire**
            si $index[k] == 0$ **alors**
                sc[i] := k ;
                $index[k] := 1$;
            fin si
        fin pour
    fin si
fin pour

3.2.3 D-HITS avec la coopération à travers la meilleure solution

La deuxième proposition distribuée est la D-HITS avec coopération de la meilleure solution que nous avons nommée DM-HITS (de l'anglais *Distributed Multi-start Hybrid Iterative Tabu Search*) (Abdelkafi et al., 2015b). Différentes RTI sont exécutées d'une manière distribuée avec différentes solutions de départ. Dans cette proposition, l'échange d'informations se fait avec la topologie en annau comme la variante DISCO-HITS. Chaque processus exécute une RTI. L'information qui est échangée pour cette variante est la meilleure solution trouvée par chaque processus. Le processus voisin reçoit l'information et exécute un UX entre sa solution courante et la meilleure solution qu'il a reçue. Plusieurs travaux de la littérature ont montré que l'utilisation de la structure de la meilleure solution trouvée permet de guider la recherche vers une zone prometteuse de l'espace de recherche (James et al., 2009a,b). Ce croisement est une diversification après chaque RT dans l'algorithme qui va permettre de générer une nouvelle solution de départ. L'UX utilisé est celui présenté dans la section précédente avec l'algorithme 12. Il utilise la séquence de sélection qui est perturbée avant chaque croisement afin de garantir la diversité des solutions générées.

Dans cette proposition, un mécanisme de mutation est implémenté mais qui est différent de la mutation classique. Dans la mutation classique, nous choisissons un individu aléatoire et nous exécutons la mutation sur une position aléatoire de cet individu. Dans notre cas, la mutation est considérée comme un mécanisme d'adaptation qui permet de continuer l'évolution. L'idée est la suivante : si un processus ne peut pas améliorer sa meilleure solution trouvée après plusieurs RT successives, cela signifie qu'il est potentiellement piégé dans un optimum local. Dans ce cas, une perturbation aléatoire est appliquée sur une partie de la solution en plus de l'UX. Cette perturbation engendre une nouvelle solution mutante pour continuer l'exploration.

L'algorithme 13 est dupliqué pour tous les processus. Une succession de RT (algorithme 9) est exécutée. Après chaque RT, l'algorithme 13 stocke la nouvelle meilleure

Algorithme 13 DM-HITS pour chaque processus

1: *Entrées* : *perturbmut* : % du taux de mutation ; *TM* : seuil pour exécuter la mutation ; *Information* : solution à échangée

2: *Sorties* : *cost* : coût de la solution courante ; *Fcost* : le meilleur coût trouvé ; *sc* : solution courante ; *Fsolution* : la meilleure solution trouvée

3: Initialisation de la solution du processus courant ;

4: $m := 0$; /* est un compteur pour vérifier la nécessité d'une mutation */

5: **répéter**

6: RT /* voir algorithme 9 */

7: **si** *cost* < *Fcost* **alors**

8: /* Amélioration */

9: *Fcost* := *cost* ; $m := 0$;

10: Mise à jour de *Fsolution* avec *sc* ;

11: **sinon**

12: m++ ;

13: **fin si**

14: Mise à jour de *Information* avec *Fsolution* ;

15: Echange de *Information* entre les processus (topologie en annau) ;

16: /* Croisement uniforme */ /* voir algorithme 12 */

17: **UX** entre *sc* et *Information* ;

18: /* Mutation */

19: **si** $m == TM$ **alors**

20: $m := 0$;

21: **Perturbation** de *sc* avec le paramètre *perturbmut* ;

22: **fin si**

23: **jusqu'à** (condition d'arrêt)

solution dans le cas d'une amélioration. Dans le cas contraire, le compteur m est incrémenté. L'étape suivante de l'algorithme est d'envoyer la meilleure solution trouvée vers un processus voisin et de recevoir la même information d'un autre voisin. C'est l'étape d'échange de données. Ensuite, l'algorithme applique un UX (algorithme 12) entre la solution courante et l'information reçue. Le résultat de ce croisement est la solution qui va servir de solution initiale à la prochaine RT. Avant d'entamer la nouvelle RT, le seuil de mutation **TM** est vérifié. Si ce seuil est atteint, l'algorithme 13 applique une perturbation sur une partie de la solution générée par le croisement.

3.2.4 Résultats et analyses

3.2.4.1 Conditions des tests

Pour l'expérimentation des variantes distribuées, les algorithmes utilisent 10 machines du cluster. L'expérimentation est appliquée sur les instances du QAPLIB (Burkard et al., 1997) ($http://www.seas.upenn.edu/qaplib/inst.html$). La taille des instances varie entre 20 et 150. Tous les résultats sont exprimés sous forme de pourcentage de déviation par rapport à la meilleure solution connue, nommée BKS (de l'anglais *Best Known Solution*) (eq 3.2). Tous les BKS peuvent être trouvées sur le site du QAPLIB. Chaque instance est exécutée 10 fois et les moyennes sont présentées dans les tableaux des résultats. Le temps est exprimé en minutes.

$$déviation = \frac{(solution - BKS) \times 100}{BKS} \tag{3.2}$$

Le QAPLIB contient 134 instances qu'on peut classifier dans quatre types :

— Les instances du monde réel (type 1) ;

— Les instances déstructurées générer aléatoirement avec une distribution uniforme (type 2) ;

— Les instances générées aléatoirement et qui sont similaires aux instances du monde réel (type 3) ;

— Les instances dans lesquelles les distances sont basées sur la distance de Manhattan sur une grille (type 4) ;

Seuls les types 2, 3 et 4 sont considérés dans ce chapitre, car le type 1 est très facile à résoudre par nos approches mais aussi par les approches de la littérature.

Nos approches contiennent un ensemble de paramètres. Ces paramètres sont fixés après un certain nombre d'expérimentations afin d'obtenir le meilleur compromis entre l'intensification et la diversification. Le tableau 3.1 propose les paramètres utilisés par nos algorithmes distribués. Le paramètre *iterRT* est le nombre d'itérations exécutées par chaque RT (algorithme 9 de la ligne 3 à 13). Le paramètre de *iterGLOBALE* est la condition d'arrêt des algorithmes et n représente le taille du problème. La valeur du *rank* est l'indice du processus courant. Le paramètre *Perturb* est le pourcentage de perturbation effectué par DISCO-HITS et *Perturbmut* est le taux de mutation effectué par DM-HITS.

TABLE 3.1 – Paramètres

Paramètres	**Valeurs**
iterRT	$1000 \times n$
iterGLOBALE	200
L1	20+rank
L2	40+rank
critère d'aspiration	$n \times n \times 5$
Perturb	25%
TM	5
Perturbmut	10%

3.2.4.2 Expérimentation des variantes distribuées

La première expérimentation (Tableau 3.2) présente les résultats des variantes distribuées. Le même nombre d'évaluations de la fonction objectif et les mêmes machines sont utilisés. La valeur entre parenthèses représente le nombre de fois où l'algorithme atteint

le BKS parmi les 10 tentatives. Les résultats sont présentés pour les types 2, 3 et 4, respectivement.

Le tableau 3.2 contient les résultats pour les quatre variantes distribuées. La première comparaison est entre D-HITS et DISCO-HITS-DG. Ces deux variantes utilisent les mêmes éléments (RT, DG, relocalisation et perturbation). La différence entre les deux variantes est le mécanisme d'échange de données entre les processus qu'utilise le DISCO-HITS-DG. Ce mécanisme donne la capacité au DISCO-HITS-DG d'explorer efficacement l'espace de recherche. Il utilise l'évolution de chaque processus pour répartir l'espace de recherche entre les processus. À travers les 34 instances, le DISCO-HITS-DG obtient de meilleurs résultats que le D-HITS sur 9 instances face à 3 pour D-HITS (tai40a, tai50a and sko100e). Le DISCO-HITS-DG a la capacité d'obtenir de meilleurs résultats pour les grandes instances telles que tai150b et tho150. Cela confirme que la méthode de coopération entre les processus est efficace pour explorer l'espace de recherche des grandes instances. Les résultats pour le type 2 sont équivalents pour les deux variantes mais DISCO-HITS-DG obtient de meilleurs résultats dans les types 3 et 4. Avec une moyenne globale de 0,046%, le DISCO-HITS-DG est meilleur que le D-HITS qui a une moyenne de 0,052%.

La deuxième comparaison est réalisée entre DISCO-HITS-DG et DISCO-HITS-UX. La différence entre ces deux variantes est seulement l'opération de diversification (la DG face à l'UX). DISCO-HITS-UX obtient la meilleure moyenne 0,038% pour les 34 instances grâce à son efficacité dans les types 2 et 4. La deuxième moyenne est pour la DISCO-HITS-DG avec 0,046% mais cette variante obtient de meilleurs résultats dans le type 3. DISCO-HITS-UX atteint le BKS 284 fois face à 279 fois pour DISCO-HITS-DG. Cela veut dire que la variante DISCO-HITS-UX est plus robuste. De plus, cette variante obtient de meilleurs résultats dans 6 instances face à seulement une seule instance pour DISCO-HITS-DG. Cette comparaison révèle l'efficacité de l'UX pour résoudre le problème du QAP par rapport à la DG sur les 34 instances expérimentées.

TABLE 3.2 – Résultats obtenus à l'aide des algorithmes distribués

Instances(34)	**BKS**	**D-HITS**		**DISCO-HITS-DG**		**DISCO-HITS-UX**		**DM-HITS**	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps
tai20a	703482	**0,000(10)**	0,41	**0,000(10)**	0,40	**0,000(10)**	0,4	**0,000(10)**	0,02
tai25a	1167256	**0,000(10)**	0,81	**0,000(10)**	0,78	**0,000(10)**	0,79	**0,000(10)**	0,04
tai30a	1818146	**0,000(10)**	1,40	**0,000(10)**	1,36	**0,000(10)**	1,37	**0,000(10)**	0,07
tai35a	2422002	**0,000(10)**	2,18	**0,000(10)**	2,16	**0,000(10)**	2,15	**0,000(10)**	0,27
tai40a	3139370	**0,007(9)**	3,25	0,030(6)	3,18	**0,007(9)**	3,22	0,015(8)	3,19
tai50a	4938796	0,058(7)	6,34	0,062(8)	6,19	0,048(8)	6,29	**0,000(10)**	6,25
tai60a	7205962	0,369(0)	11,10	0,303(0)	10,71	**0,272(0)**	10,77	0,319(0)	10,73
tai80a	13515450	0,654(0)	26,58	0,573(0)	25,54	0,561(0)	25,62	**0,496(0)**	25,81
tai100a	21052466	0,582(0)	54,30	0,552(0)	52,07	**0,359(0)**	52,36	0,563(0)	52,20
tai20b	122455319	**0,000(10)**	0,31	**0,000(10)**	0,18	**0,000(10)**	0,18	**0,000(10)**	0,00
tai25b	344355646	**0,000(10)**	0,75	**0,000(10)**	0,7	**0,000(10)**	0,69	**0,000(10)**	0,01
tai30b	637117113	**0,000(10)**	1,36	**0,000(10)**	1,34	**0,000(10)**	1,36	**0,000(10)**	0,14
tai35b	283315445	**0,000(10)**	2,14	**0,000(10)**	2,13	**0,000(10)**	2,11	**0,000(10)**	0,09
tai40b	637250948	**0,000(10)**	3,18	**0,000(10)**	3,17	**0,000(10)**	3,16	**0,000(10)**	0,28
tai50b	458821517	**0,000(10)**	6,19	**0,000(10)**	6,12	**0,000(10)**	6,09	**0,000(10)**	1,08
tai60b	608215054	**0,000(10)**	10,64	**0,000(10)**	10,67	**0,000(10)**	10,6	**0,000(10)**	5,93
tai80b	818415043	**0,000(10)**	25,36	**0,000(10)**	25,60	**0,000(10)**	25,35	**0,000(10)**	21,10
tai100b	1185996137	0,000(6)	52,83	0,000(9)	51,26	**0,000(10)**	53,08	**0,000(10)**	51,32
tai150b	498896643	0,076(0)	192,48	0,015(0)	214,26	0,027(3)	214,86	**0,000(10)**	208,74
sko42	15812	**0,000(10)**	3,68	**0,000(10)**	3,67	**0,000(10)**	3,25	**0,000(10)**	0,45
sko49	23386	**0,000(10)**	5,80	**0,000(10)**	5,76	**0,000(10)**	2,67	**0,000(10)**	2,60
sko56	34458	**0,000(10)**	8,66	**0,000(10)**	8,63	**0,000(10)**	8,59	**0,000(10)**	4,60
sko64	48498	**0,000(10)**	12,99	**0,000(10)**	12,94	**0,000(10)**	7	**0,000(10)**	6,80
sko72	66256	**0,000(10)**	18,66	**0,000(10)**	18,53	**0,000(10)**	18,53	**0,000(10)**	12,70
sko81	90998	0,001(9)	26,56	**0,000(10)**	26,48	**0,000(10)**	26,62	**0,000(10)**	24,81
sko90	115534	**0,000(10)**	37,05	**0,000(10)**	37,17	**0,000(10)**	37,24	**0,000(10)**	29,56
sko100a	152002	0,001(9)	51,37	**0,000(10)**	53,65	**0,000(10)**	52,41	**0,000(10)**	43,01
sko100b	153890	**0,000(10)**	52,04	**0,000(10)**	51,50	**0,000(10)**	52,63	**0,000(10)**	48,76
sko100c	147862	0,000(7)	52,76	**0,000(10)**	51,52	**0,000(10)**	52,04	**0,000(10)**	47,07
sko100d	149576	0,001(6)	52,48	**0,000(10)**	54,09	0,000(8)	51,28	**0,000(10)**	38,97
sko100e	149150	**0,000(10)**	53,27	0,001(9)	51,42	0,001(9)	51,30	**0,000(10)**	42,87
sko100f	149036	0,001(9)	52,36	0,001(8)	51,48	0,001(9)	51,64	**0,000(10)**	42,15
wil100	273038	0,002(0)	53,57	0,000(9)	51,60	0,000(8)	51,62	**0,000(10)**	37,76
tho150	8133398	0,027(0)	217,34	0,010(0)	218,95	0,004(0)	199,36	**0,001(9)**	194,88
Moyenne type 2		0,186(56)	11,81	0,186(54)	11,38	**0,139(57)**	11,44	0,155(58)	10,95
Moyenne type 3		0,008(86)	29,52	0,002(89)	31,54	0,003(93)	31,75	**0,000(100)**	28,87
Moyenne type 4		0,002(120)	46,57	0,001(136)	46,49	0,000(134)	44,41	**0,000(149)**	38,47
Moyenne totale		0,052(262)	32,36	0,046(279)	32,80	**0,038(284)**	31,96	0,041(307)	28,36

La dernière comparaison est entre DISCO-HITS-UX et DM-HITS. La différence entre ces deux variantes est la méthode d'échange d'informations. L'échange de données utilisées par DISCO-HITS-UX est orienté vers l'exploration de l'espace de recherche. L'estimation de la distance a pour but d'évaluer si deux solutions sont proches dans l'espace de recherche. Cette information permet à notre algorithme de prendre les décisions qui permettent de relocaliser une des deux solutions afin d'avoir une bonne répartition dans l'exploration de l'espace de recherche. Pour le DM-HITS, l'échange de données se base sur un croisement avec la meilleure solution et donc il est orienté vers l'intensification de la recherche. DISCO-HITS-UX obtient la meilleure moyenne 0,038% pour les 34 instances grâce à son efficacité dans le type 2. La deuxième moyenne est pour la DM-HITS avec 0,041% mais cette variante obtient de meilleurs résultats dans les types 3 et 4. DISCO-HITS-UX atteint le BKS 284 fois face à 304 fois pour DM-HITS. Cela veut dire que la variante DM-HITS est plus robuste. De plus, cette variante obtient de meilleurs résultats pour 6 instances face à 3 instances pour DISCO-HITS-UX. Chaque méthode d'échange de données a ses propres atouts. L'utilisation des distances dans l'échange de données améliore significativement les performances pour les instances de type 2 (instances déstructurées). Par contre, l'échange de données qui utilise les meilleures solutions trouvées est plus robuste et plus homogène sur la majorité des instances testées.

3.2.4.3 Comparaison du DISCO-HITS avec la littérature

La première comparaison avec la littérature est réalisée avec les variantes DISCO-HITS-UX et DISCO-HITS-DG. Le tableau 3.3, le tableau 3.4 et le tableau 3.5 présentent une comparaison avec quatre propositions de la littérature.

— BMA (de l'anglais *Breakout Memetic Algorithm*) (Benlic & Hao, 2015) ;
— BLS (de l'anglais *Breakout Local Search*) (Benlic & Hao, 2013) ;
— CPTS (de l'anglais *Cooperative Parallel Tabu Search*) (James et al., 2009a) ;
— PILS (de l'anglais *Population-based Iterated Local Search*) (Stützle, 2006) ;

Les deux meilleurs algorithmes de la littérature en termes de qualité de solution sont BMA et BLS. Ils utilisent le temps comme critère d'arrêt et ne donne pas le nombre d'appels de la fonction objectif par leurs algorithmes. Pour une comparaison équitable, nous avons fait en sorte que nos variantes ne dépassent pas les limites imposés par ces deux algorithmes, soit 1 heure de calcul pour les instances de taille $n <= 100$ et 4 heures de calcul pour les instances de taille $n > 100$.

Nos variantes distribuées utilisent, comme indiqué dans le tableau 3.1, $10 \times 1000 \times n \times 200$ appels à la fonction objectif, i.e. 10 processus qui lancent 200 RT où chaque RT exécute $1000 \times n$ itérations. CPTS est le troisième meilleur algorithme auquel nous nous comparons. Il utilise $10 \times 50 \times n \times 200 \times n$ appels à la fonction objectif, i.e. 10 processus qui lancent $200 \times n$ RT où chaque RT exécute $50 \times n$ itérations. En moyenne , nous utilisons 5 fois moins d'appels à la fonction objectif que CPTS.

La comparaison est surtout centrée sur la qualité des solutions proposées. Les tests sont effectués sur les 34 instances les plus difficiles du QAPLIB. Les autres instances sont faciles à résoudre pour nos algorithmes et ceux de la littérature à l'exception de l'instance tai256c.

Le tableau 3.3 présente l'expérimentation du type 2. La meilleure moyenne est obtenue par l'algorithme BMA avec 0,1294%. Cet algorithme obtient néanmoins un résultat proche de celui de DISCO-HITS-UX avec une moyenne de 0,1386%. De surcroît, le DISCO-HITS-UX obtient de meilleurs résultats que BMA pour trois instances (tai40a, tai50a et tai100a) contre deux (tai60a et tai80a). DISCO-HITS-UX surpasse les trois autres algorithmes de la littérature également (BLS, CPTS, PILS). Notre deuxième proposition qui est DISCO-HITS-DG est classée à la quatrième place après BMA, DISCO-HITS-UX et BLS.

TABLE 3.3 – Comparaison de DISCO-HITS-DG et DISCO-HITS-UX avec BMA, BLS, CPTS et PILS (type 2)

Instances(9)	BKS	DISCO-HITS-DG		DISCO-HITS-UX		BMA		BLS		CPTS		PILS	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps
tai20a	703482	**0,000(10)**	0,40	**0,000(10)**	0,4	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	0,1	**0,000(10)**	0
tai25a	1167256	**0,000(10)**	0,78	**0,000(10)**	0,79	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	0,3	**0,000(10)**	0,2
tai30a	1818146	**0,000(10)**	1,36	**0,000(10)**	1,37	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	1,6	**0,000(10)**	0,6
tai35a	2422002	**0,000(10)**	2,16	**0,000(10)**	2,15	**0,000(10)**	-	**0,000(10)**	0,2	**0,000(10)**	2,3	**0,000(10)**	2,3
tai40a	3139370	0,030(6)	3,18	**0,007(9)**	3,22	0,059(2)	8,1	0,022(7)	38,9	0,148(1)	3,5	0,280(0)	12,0
tai50a	4938796	0,062(8)	6,19	**0,048(8)**	6,29	0,131(2)	42,0	0,157(2)	45,1	0,440(0)	10,3	0,663(0)	11,2
tai60a	7205962	0,303(0)	10,71	0,272(0)	10,77	**0,144(2)**	67,5	0,251(1)	47,9	0,476(0)	26,4	0,820(0)	7,4
tai80a	13515450	0,573(0)	25,54	0,561(0)	25,62	**0,426(0)**	65,8	0,517(0)	47,3	0,691(0)	94,8	0,927(0)	12,7
tai100a	21052466	0,552(0)	52,07	**0,359(0)**	52,36	0,405(0)	44,1	0,430(0)	39,0	0,589(0)	261,2	1,027(0)	9,8
Moyenne type 2		0,1863(54)	11,38	0,1386(57)	11,44	**0,1294(46)**	-	0,1530(50)	24,27	0,2604(41)	44,50	0,4130(40)	6,24

Le tableau 3.4 présente l'expérimentation du type 3. La meilleure moyenne est obtenue pour notre algorithme DISCO-HITS-DG avec 0,0015%. DISCO-HITS-UX obtient, quant à lui, une valeur égale à 0,0027%. L'instance la plus difficile à résoudre du type 3 est la tai150b.

TABLE 3.4 – Comparaison de DISCO-HITS-DG et DISCO-HITS-UX avec BMA, BLS, CPTS et PILS (type 3)

Instances(10)	BKS	DISCO-HITS-DG		DISCO-HITS-UX		BMA		BLS		CPTS		PILS	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps
tai20b	122455319	**0,000(10)**	0,18	**0,000(10)**	0,18	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	0,1	**0,000(10)**	0
tai25b	344355646	**0,000(10)**	0,7	**0,000(10)**	0,69	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	0,4	**0,000(10)**	0
tai30b	637117113	**0,000(10)**	1,34	**0,000(10)**	1,36	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	1,2	**0,000(10)**	0
tai35b	283315445	**0,000(10)**	2,13	**0,000(10)**	2,11	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	2,4	**0,000(10)**	0
tai40b	637250948	**0,000(10)**	3,17	**0,000(10)**	3,16	**0,000(10)**	-	**0,000(10)**	0	**0,000(10)**	4,5	**0,000(10)**	0
tai50b	458821517	**0,000(10)**	6,12	**0,000(10)**	6,09	**0,000(10)**	1,2	**0,000(10)**	2,8	**0,000(10)**	13,8	**0,000(10)**	0,1
tai60b	608215054	**0,000(10)**	10,67	**0,000(10)**	10,6	**0,000(10)**	5,2	**0,000(10)**	5,6	**0,000(10)**	30,4	**0,000(10)**	0,2
tai80b	818415043	**0,000(10)**	25,6	**0,000(10)**	25,35	**0,000(10)**	31,3	**0,000(10)**	11,4	**0,000(10)**	110,9	**0,000(10)**	1,3
tai100b	1185996137	0,000(9)	51,26	**0,000(10)**	53,08	**0,000(10)**	13,6	**0,000(10)**	16,0	0,001(8)	241,0	**0,000(10)**	2,3
tai150b	498896643	**0,015(0)**	214,26	0,027(3)	214,86	0,060(1)	78,1	0,075(1)	243,6	0,076(0)	7377,8	0,095(0)	36,7
Moyenne type 3		**0,0015(89)**	31,54	0,0027(93)	31,75	0,006(91)	-	0,0075(88)	27,94	0,0077(88)	778,25	0,0095(90)	4,06

Le tableau 3.5 présente l'expérimentation du type 4. La meilleure moyenne est réalisée par l'algorithme DISCO-HITS-UX avec 0,0004%. Il est suivi par notre deuxième variante qui est le DISCO-HITS-DG avec un résultat moyen de 0,0008%. L'instance la plus difficile

à résoudre de ce type est la tho150. Dans cette comparaison, les deux algorithmes que nous proposons obtiennent les meilleurs résultats. Ils montrent une très bonne efficacité à résoudre les grandes instances de taille 150.

TABLE 3.5 – Comparaison de DISCO-HITS-DG et DISCO-HITS-UX avec BMA, BLS, CPTS et PILS (type 4)

Instances(15)	**BKS**	**DISCO-HITS-DG**		**DISCO-HITS-UX**		**BMA**		**BLS**		**CPTS**		**PILS**	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps	déviation	temps
sko42	15812	**0,000(10)**	3,67	**0,000(10)**	3,25	**0,000(10)**	-	**0,000(10)**	1,7	**0,000(10)**	5,3	**0,000(10)**	2,8
sko49	23386	**0,000(10)**	5,76	**0,000(10)**	2,67	**0,000(10)**	-	**0,000(10)**	0,5	**0,000(10)**	11,4	**0,000(10)**	0,8
sko56	34458	**0,000(10)**	8,63	**0,000(10)**	8,59	**0,000(10)**	-	**0,000(10)**	1,1	**0,000(10)**	21	**0,000(10)**	0,5
sko64	48498	**0,000(10)**	12,94	**0,000(10)**	7	**0,000(10)**	-	**0,000(10)**	1,3	**0,000(10)**	42,9	**0,000(10)**	0,2
sko72	66256	**0,000(10)**	18,53	**0,000(10)**	18,53	**0,000(10)**	3,5	**0,000(10)**	4,1	**0,000(10)**	69,6	0,001(8)	6,7
sko81	90998	**0,000(10)**	26,48	**0,000(10)**	26,62	**0,000(10)**	4,3	**0,000(10)**	13,9	**0,000(10)**	121,4	0,007(5)	9,6
sko90	115534	**0,000(10)**	37,17	**0,000(10)**	37,24	**0,000(10)**	15,3	**0,000(10)**	16,6	**0,000(10)**	193,7	0,006(4)	10,6
sko100a	152002	**0,000(10)**	53,65	**0,000(10)**	52,41	**0,000(10)**	22,3	0,001(9)	20,8	**0,000(10)**	304,8	0,012(3)	7,9
sko100b	153890	**0,000(10)**	51,5	**0,000(10)**	52,63	**0,000(10)**	6,5	**0,000(10)**	10,8	**0,000(10)**	309,6	0,007(5)	7,3
sko100c	147862	**0,000(10)**	51,52	**0,000(10)**	52,04	**0,000(10)**	12,0	**0,000(10)**	15,5	**0,000(10)**	316,1	0,002(6)	11,5
sko100d	149576	**0,000(10)**	54,09	0,000(8)	51,28	0,006(9)	20,9	0,001(5)	38,9	**0,000(10)**	309,8,6	0,021(0)	11,8
sko100e	149150	0,001(9)	51,42	0,001(9)	51,30	**0,000(10)**	11,9	**0,000(10)**	42,5	**0,000(10)**	309,1	0,001(7)	6,8
sko100f	149036	0,001(8)	51,48	0,001(9)	51,64	**0,000(10)**	23,0	**0,000(10)**	17,3	0,003(4)	310,3	0,037(0)	11,7
wil100	273038	0,000(9)	51,60	0,000(8)	51,62	**0,000(10)**	14,5	**0,000(10)**	18,9	**0,000(10)**	316,6	0,004(1)	6,3
tho150	8133398	0,010(0)	218,95	**0,004(0)**	199,36	0,008(3)	416,4	0,023(1)	268,8	0,013(0)	1991,7	0,068(0)	36,2
Moyenne type 4		0,0008(136)	46,49	**0,0004(134)**	44,41	0,0009(142)	-	0,0016(135)	31,51	0,0011(134)	308,89	0,0111(79)	8,71

3.2.4.4 Comparaison du DM-HITS avec la littérature

La deuxième comparaison avec la littérature est réalisée pour comparer le DM-HITS. Les mêmes conditions expérimentales que dans la première comparaison sont utilisées. La comparaison est séparée en deux tableaux (tableau 3.6 et tableau 3.7). Ces deux tableaux présentent l'expérimentation du DM-HITS avec les quatre algorithmes déjà cités dans la première comparaison en plus de deux autres algorithmes qui sont :

— ITS (de l'anglais *Iterative Tabu search*) (Misevicius & Kilda, 2006) ;

— IHGA (de l'anglais *Improved Hybrid Genetic Algorithm*) (Misevicius, 2004).

Le tableau 3.6 propose une comparaison avec BLS, CPTS et PILS. La moyenne globale du DM-HITS, pour les 34 instances expérimentées, est meilleure que celle des trois

TABLE 3.6 – Comparaison de DM-HITS avec BLS, CPTS et PILS

Instances(34)	**BKS**	**DM-HITS**		**BLS**		**CPTS**		**PILS**	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps
tai20a	703482	**0,000(10)**	0,02	**0,000(10)**	0,0	**0,000(10)**	0,1	**0,000(10)**	0,0
tai25a	1167256	**0,000(10)**	0,04	**0,000(10)**	0,0	**0,000(10)**	0,3	**0,000(10)**	0,2
tai30a	1818146	**0,000(10)**	0,07	**0,000(10)**	0,0	**0,000(10)**	1,6	**0,000(10)**	0,6
tai35a	2422002	**0,000(10)**	0,27	**0,000(10)**	0,2	**0,000(10)**	2,3	**0,000(10)**	2,3
tai40a	3139370	**0,015(8)**	3,19	0,022(7)	38,9	0,148(1)	3,5	0,280(0)	12,0
tai50a	4938796	**0,000(10)**	6,25	0,157(2)	45,1	0,440(0)	10,3	0,663(0)	11,2
tai60a	7205962	0,319(0)	10,73	**0,251(1)**	47,9	0,476(0)	26,4	0,820(0)	7,4
tai80a	13515450	**0,496(0)**	25,81	0,517(0)	47,3	0,691(0)	94,8	0,927(0)	12,7
tai100a	21052466	0,563(0)	52,20	**0,430(0)**	39,0	0,589(0)	261,2	1,027(0)	9,8
tai20b	122455319	**0,000(10)**	0,00	**0,000(10)**	0,0	**0,000(10)**	0,1	**0,000(10)**	0,0
tai25b	344355646	**0,000(10)**	0,01	**0,000(10)**	0,0	**0,000(10)**	0,4	**0,000(10)**	0,0
tai30b	637117113	**0,000(10)**	0,14	**0,000(10)**	0,0	**0,000(10)**	1,2	**0,000(10)**	0,0
tai35b	283315445	**0,000(10)**	0,09	**0,000(10)**	0,0	**0,000(10)**	2,4	**0,000(10)**	0,0
tai40b	637250948	**0,000(10)**	0,28	**0,000(10)**	0,0	**0,000(10)**	4,5	**0,000(10)**	0,0
tai50b	458821517	**0,000(10)**	1,08	**0,000(10)**	2,8	**0,000(10)**	13,8	**0,000(10)**	0,1
tai60b	608215054	**0,000(10)**	5,93	**0,000(10)**	5,6	**0,000(10)**	30,4	**0,000(10)**	0,2
tai80b	818415043	**0,000(10)**	21,10	**0,000(10)**	11,4	**0,000(10)**	110,9	**0,000(10)**	1,3
tai100b	1185996137	**0,000(10)**	51,32	**0,000(10)**	16,0	0,001(8)	241,0	**0,000(10)**	2,3
tai150b	498896643	**0,000(10)**	208,74	0,075(1)	243,6	0,076(0)	7377,8	0,095(0)	36,7
sko42	15812	**0,000(10)**	0,45	**0,000(10)**	1,7	**0,000(10)**	5,3	**0,000(10)**	2,8
sko49	23386	**0,000(10)**	2,60	**0,000(10)**	0,5	**0,000(10)**	11,4	**0,000(10)**	0,8
sko56	34458	**0,000(10)**	4,60	**0,000(10)**	1,1	**0,000(10)**	21,0	**0,000(10)**	0,5
sko64	48498	**0,000(10)**	6,80	**0,000(10)**	1,3	**0,000(10)**	42,9	**0,000(10)**	0,2
sko72	66256	**0,000(10)**	12,70	**0,000(10)**	4,1	**0,000(10)**	69,6	0,001(8)	6,7
sko81	90998	**0,000(10)**	24,81	**0,000(10)**	13,9	**0,000(10)**	121,4	0,007(5)	9,6
sko90	115534	**0,000(10)**	29,56	**0,000(10)**	16,6	**0,000(10)**	193,7	0,006(4)	10,6
sko100a	152002	**0,000(10)**	43,01	0,001(9)	20,8	**0,000(10)**	304,8	0,012(3)	7,9
sko100b	153890	**0,000(10)**	48,76	**0,000(10)**	10,8	**0,000(10)**	309,6	0,007(5)	7,3
sko100c	147862	**0,000(10)**	47,07	**0,000(10)**	15,5	**0,000(10)**	316,1	0,002(6)	11,5
sko100d	149576	**0,000(10)**	38,97	0,001(5)	38,9	**0,000(10)**	309,8,6	0,021(0)	11,8
sko100e	149150	**0,000(10)**	42,87	**0,000(10)**	42,5	**0,000(10)**	309,1	0,001(7)	6,8
sko100f	149036	**0,000(10)**	42,15	**0,000(10)**	17,3	0,003(4)	310,3	0,037(0)	11,7
wil100	273038	**0,000(10)**	37,76	**0,000(10)**	18,9	**0,000(10)**	316,6	0,004(1)	6,3
tho150	8133398	**0,001(9)**	194,88	0,023(1)	268,8	0,013(0)	1991,7	0,068(0)	36,2
Moyenne totale		**0,041(307)**	28,36	0,043(273)	28,5	0,072(263)	377,0	0,117(209)	6,7
Moyenne type 2		0,155(58)	10,95	**0,153(50)**	24,3	0,260(41)	44,5	0,413(40)	6,2
Moyenne type 3		**0,000(100)**	28,87	0,008(88)	27,9	0,008(88)	778,3	0,010(90)	4,1
Moyenne type 4		**0,000(149)**	38,47	0,002(135)	31,5	0,001(134)	308,8	0,011(79)	8,7

approches comparées. DM-HITS obtient une moyenne de 0,041% face à 0,043% pour BLS qui est la meilleure approche parmi les 3 de la littérature dans le tableau 3.6. Pour les 34 instances, DM-HITS n'obtient de moins bons résultats que sur 2 instances. Pour ces deux instances, notre proposition est classée deuxième derrière BLS. La variante DM-HITS est très compétitive sur les types 3 et 4. De très difficiles instances telles que tai40a, tai50a, tai80a, tai150b et tho150 sont résolues efficacement en comparaison avec la littérature. Notre proposition surpasse BLS sur 7 instances (sko100a, sko100d, tho150, tai40a, tai50a, tai80a et tai150b). Elle est classée en seconde position seulement sur les deux instances déjà évoquées qui sont tai60a et tai100a. Le BLS obtient de meilleurs résultats sur le type 2 mais la différence de moyenne par rapport à notre variante est de 0,002%, i.e. 0,155% pour DM-HITS face à 0,153% pour BLS.

Dans le deuxième tableau de la comparaison (tableau 3.7), le DM-HITS est comparé avec un des meilleurs algorithmes de la littérature pour résoudre le QAP qui est BMA. La comparaison est réalisée avec deux autres algorithmes qui sont l'IHGA et l'ITS. BMA et DM-HITS obtiennent tout les deux le BKS dans les 10 tentatives pour 26 instances parmit les 34 instances testées du QAPLIB. La comparaison est réalisée uniquement sur les 8 instances restantes, là où DM-HITS ou BMA n'arrivent pas à atteindre le BKS dans toutes les tentatives. Le DM-HITS surpasse BMA pour 5 instances qui sont tai40a, tai50a, sko100d, tho150 et tai150b contre 3 instances pour BMA qui sont tai60a, tai80a et tai100a.

Il est à noter qu'en termes de moyenne globale des 34 instances proposées dans ce chapitre, le BMA obtient la meilleure moyenne de 0,036% contre 0,041% pour DM-HITS ou encore 0,038% pour DISCO-HITS-UX. Le DM-HITS surpasse le BMA dans les types 3 et 4. Il obtient 0% dans le type 3 contre 0,006% pour BMA. Pour le type 4, DM-HITS obtient 0,0001% contre 0,0009% pour le BMA. Pour le type 2, le BMA arrive à se rattraper face à DM-HITS avec une moyenne de 0,129% contre 0,155% pour le DM-HITS.

TABLE 3.7 – Comparaison de DM-HITS avec BMA, ITS et IHGA

Instances(8)	BKS	DM-HITS		BMA		ITS		IHGA	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps
sko100d	149576	**0,000(10)**	38,97	0,006(9)	20,9	-	-	-	-
tho150	8133398	**0,001(9)**	194,88	0,008(3)	416,4	-	-	-	-
tai40a	3139370	**0,015(8)**	3,19	0,059(2)	8,1	0,210(1)	0,8	0,209(1)	1,4
tai50a	4938796	**0,000(10)**	6,25	0,131(2)	42,0	0,373(0)	3,0	0,262(0)	5,0
tai60a	7205962	0,319(0)	10,73	**0,144(2)**	67,5	0,330(1)	9,7	0,583(0)	12
tai80a	13515450	0,496(0)	25,81	**0,426(0)**	65,8	0,494(0)	25,0	0,756(0)	53,3
tai100a	21052466	0,563(0)	52,20	**0,405(0)**	44,1	0,427(0)	60,0	0,606(0)	200,0
tai150b	498896643	**0,000(10)**	208,74	0,060(1)	78,1	0,100(1)	60,0	0,111(2)	38,3

3.3 Niveau de parallélisation des structures de voisinage

Dans cette partie du chapitre 3, nous proposons d'ajouter **le niveau de parallélisation des structures de voisinage** avec GPU (Abdelkafi et al., 2015c). L'objectif est de paralléliser l'évaluation du voisinage de la RT utilisée dans nos variantes distribuées. Comme vu précédemment, chaque HITS est exécuté sur un processus. Le HITS exécute plusieurs RT successives séparées par des diversifications adaptatives.

Pour évaluer le mouvement de permutation, notre RT utilise une matrice nommée matrice delta qui stocke le coût associé à toutes les possibilités de permutation. Ces nouveaux coûts calculés peuvent alors être ajoutés aux coûts originaux des permutations pour obtenir les coûts de tous les voisins possibles avec les permutations à partir d'une solution courante. Ce calcul permet de réduire la complexité de l'évaluation des voisins. Plus d'informations sur l'utilisation de la fonction objectif de la matrice delta peuvent

être trouvées dans les travaux de Taillard (1991), James et al. (2009b) et Benlic & Hao (2013).

La partie la plus coûteuse en temps de calcul de nos approches distribuées est l'évaluation du voisinage dans la RT. Dans notre proposition de parallélisation sur GPU, chaque thread représente un mouvement de permutation. En effet, tous les mouvements d'évaluation d'une permutation sont parallèles afin d'utiliser au mieux la puissance de calcul du GPU. La première étape est de créer une matrice delta à une seule dimension directement sur le GPU pour éviter un transfert supplémentaire entre CPU et GPU. La taille de la matrice delta varie avec la taille de l'instance. Elle est de taille $n \times n$ avec n la taille de l'instance.

Seul deux kernels sont créés pour l'évaluation des voisins en parallèle. Le premier kernel (*initialisation de delta*) est utilisé dans l'initialisation des premiers coûts de voisinage dans la matrice. Le deuxième kernel (*mise à jour de delta*) est utilisé dans la mise à jour de la matrice à chaque itération de la RT. L'algorithme 14 et l'algorithme 15 représentent respectivement le kernel d'initialisation de delta et de mise à jour de delta.

Puisque nous utilisons une matice delta à une seule dimension, nous devons récupérer les indices de représentation dans une matrice à deux dimensions pour calculer la permutation dans chaque thread (ligne 3 et 4 dans l'algorithme 14 et l'algorithme 15). La variable z (ligne 6 dans l'algorithme 14) calcule le nouveau coût du mouvement de permutation pour le thread courant. L'algorithme 14 et l'algorithme 15 sont exécutés en parallèle par tous les threads. Pour une instance de taille n nous générons $n \times n$ threads.

Ces deux kernels ont besoin de transfert d'informations entre le host et le device. Avant chaque exécution de ces kernels, la solution courante *sc* (ligne 1 algorithme 14) est transférée du host vers le device pour qu'elle soit utilisée dans le calcul de la nouvelle matrice delta. Après l'exécution des kernels, la matrice delta est transférée du device vers le host pour trouver la meilleure permutation et générer le bon voisin.

Algorithme 14 Kernel d'initialisation de delta

1: *Entrées* : T : la taille de l'instance ; sc : la solution courante ; fl et d : les matrices de flux et de distances respectivement
2: *Sortie* : *delta* : la matrice delta
3: Obtenir l'indice du thread idx ; /* chaque idx représente un mouvement de permutation */
4: $i := \lfloor idx/T \rfloor$;
5: $j := idx \bmod T$; /* i et j représentent les indices qui permettent le calcul des mouvements */
6: **si** $i < j$ **alors**
7: $z := (fl[i \times T+i] - fl[j \times T+j]) \times (d[sc[j] \times T+sc[j]] - d[sc[i] \times T+sc[i]]) + (fl[i \times T+j] - fl[j \times T+i]) \times (d[sc[j] \times T+sc[i]] - d[sc[i] \times T+sc[j]])$;
8: **pour** $k := 0$ à T **faire**
9: **si** $k \neq i$ **et** $k \neq j$ **alors**
10: $z := z + (fl[k \times T+i] - fl[k \times T+j]) \times (d[sc[k] \times T+sc[j]] - d[sc[k] \times T+sc[i]]) + (fl[i \times T+k] - fl[j \times T+k]) \times (d[sc[j] \times T+sc[k]] - d[sc[i] \times T+sc[k]])$;
11: **fin si**
12: **fin pour**
13: $delta[idx] := z$;
14: **fin si**

Algorithme 15 Kernel de mise à jour de delta

1: *Entrées* : T : la taille de l'instance ; sc : la solution courante ; fl et d : les matrices de flux et de distances respectivement ; *iretained*, *jretained* : les indices qui représentent le meilleur mouvement
2: *Sortie* : *delta* : la matrice delta
3: Obtenir l'indice du thread idx ; /* chaque idx représente un mouvement de permutation */
4: i := $\lfloor idx/T \rfloor$;
5: j := idx mod T ;/* i and j représentent les indices qui permettent le calcul des mouvements */
6: **si** $i < j$ **alors**
7: **si** $i \neq iretained$ **et** $i \neq jretained$ **et** $j \neq iretained$ **et** $j \neq jretained$ **alors**
8: z := *delta*[*idx*] + (*fl*[*iretained*× T+*i*] - *fl*[*iretained*× T+*j*] + *fl*[*jretained*× T+*j*] - *fl*[*jretained*× T+*i*]) × (*d*[*sc*[*jretained*]× T+*sc*[*i*]] - *d*[*sc*[*jretained*]× T+*sc*[*j*]] + *d*[*sc*[*iretained*]× T+*sc*[*j*]] - *d*[*sc*[*iretained*]× T+*sc*[*i*]]) + (*fl*[*i*× T+*iretained*] - *fl*[*j*× T+*iretained*] + *fl*[*j*× T+*jretained*] - *fl*[*i*× T+*jretained*]) × (*d*[*sc*[*i*]× T+*sc*[*jretained*]] - *d*[*sc*[*j*]× T+*sc*[*jretained*]] + *d*[*sc*[*j*]× T+*sc*[*iretained*]] - *d*[*sc*[*i*]× T+*sc*[*iretained*]]) ;
9: **sinon**
10: z := (*fl*[*i*× T+*i*] - *fl*[*j*× T+*j*]) × (*d*[*sc*[*j*]× T+*sc*[*j*]] - *d*[*sc*[*i*]× T+*sc*[*i*]]) + (*fl*[*i*× T+*j*] - *fl*[*j*× T+*i*]) × (*d*[*sc*[*j*]× T+*sc*[*i*]] - *d*[*sc*[*i*]× T+*sc*[*j*]]) ;
11: **pour** k := 0 à T **faire**
12: **si** $k \neq i$ **et** $k \neq j$ **alors**
13: z := z + (*fl*[*k*× T+*i*] - *fl*[*k*× T+*j*]) × (*d*[*sc*[*k*]× T+*sc*[*j*]] - *d*[*sc*[*k*]× T+*sc*[*i*]]) + (*fl*[*i*× T+*k*] - *fl*[*j*× T+*k*]) × (*d*[*sc*[*j*]× T+*sc*[*k*]] - *d*[*sc*[*i*]× T+*sc*[*k*]]) ;
14: **fin si**
15: **fin pour**
16: **fin si**
17: *delta*[*idx*] := z ;
18: **fin si**

3.3.1 Conditions de tests

L'expérimentation de cette parallélisation est exécutée sur les instances du QAPLIB et les instances proposées par Drezner et al. (2005) (*http ://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html*). La taille des instances varie entre 20 et 343. Tous les résultats sont exprimés sous forme de pourcentage de déviation par rapport au BKS (eq 3.2). Chaque instance est exécutée 10 fois et la moyenne des résultats est présentée dans les tableaux des résultats.

Deux expérimentations sont effectuées. La première expérimentation est sur les instances de taille entre 20 et 343 pour déterminer la taille d'instance à partir de laquelle notre algorithme parallèle montre une accélération significative du temps de calcul. La deuxième expérimentation est destinée aux grandes instances de taille 343. L'algorithme D-HITS est exécuté sur un seul processus (ce qui revient à une version séquentielle). Ensuite, le même algorithme est expérimenté avec une RT parallèle sur GPU. Le tableau 3.8 présente les paramètres de la première simulation et le tableau 3.9 présente ceux de la deuxième simulation.

TABLE 3.8 – Paramètres de la simulation sur les instances de taille inférieure à 256

Paramètres	**Valeurs**
iterRT	$500 \times n$
iterGLOBALE	100
L1	20
L2	40
critère d'aspiration	$n \times n \times 5$
pourcentage de perturbation	25%

TABLE 3.9 – Paramètres de la simulation des instances de taille 343

Paramètres	**Valeurs**
Nombre de processus	6
iterRT	$1000 \times n$
iterGLOBALE	10
L1	20+rank
L2	40+rank
critère d'aspiration	$n \times n \times 5$
Perturb	25%
TM	5
taux de mutation	10%

3.3.2 Simulation de la parallélisation GPU

La première simulation est présentée dans le tableau 3.10. La parallélisation est comparée avec les trois meilleures variantes du travail de James et al. (2009b).

Le critère d'arrêt mentionné dans l'article de James et al. (2009b) est de $50000 \times n$. Pour cette raison, nous utilisons dans notre simulation le même nombre d'appels de la fonction objectif qui est dans les paramètres du tableau 3.8. Le temps de calcul dans le tableau 3.10 est exprimé en secondes. L'algorithme qui utilise le GPU sur un seul processus est nommé $D-HITS_{GPU}$ et celui qui est totalement séquentiel est nommé $D-HITS_{CPU}$.

La même qualité de solution est obtenue entre la $D-HITS_{GPU}$ et le $D-HITS_{CPU}$. Dans le tableau 3.10, en utilisant le même nombre d'appels à la fonction objectif, la moyenne totale pour les 31 instances de l'exécution massivement parallèle surpasse tous les moyennes globales des variantes de James et al. (2009b) pour les instances testées. La moyenne globale de D-HITS massivement parallèle est de 0,103% face à 0,112% pour divTS qui représente la meilleure variante de James et al. (2009b).

Le tableau 3.11 présente le temps d'exécution CPU et celui du GPU noté $temps_{CPU}$ et $temps_{GPU}$, respectivement. L'accélération entre CPU et GPU, calculée par $\frac{GPU}{CPU}$, est notée ACC.

TABLE 3.10 – Nos résultats comparés avec ceux obtenus par (James et al., 2009b)

Instances	**BKS**	$D-HITS_{GPU}$		TTMTS		BSFTS		divTS	
		déviation	temps	déviation	temps	déviation	temps	déviation	temps
tai20a	703482	**0,000(10)**	0,393	0,000	0,23	0,000	0,24	0,000	0,24
tai25a	1167256	**0,000(10)**	0,649	0,000	0,50	0,000	0,51	0,000	0,56
tai30a	1818146	**0,000(10)**	1,965	0,000	1,41	0,000	1,51	0,000	1,31
tai35a	2422002	**0,000(10)**	5,364	0,056	3,16	0,073	3,60	0,000	4,44
tai40a	3139370	**0,099(0)**	190,656	0,284	5,22	0,311	4,81	0,222	5,16
tai50a	4938796	**0,597(0)**	302,625	0,700	10,07	0,685	10,46	0,725	10,23
tai60a	7205962	**0,623(0)**	451,616	0,820	25,92	0,752	20,65	0,718	25,69
tai80a	13515450	0,782(0)	848,055	0,817	69,21	0,841	54,61	**0,753**	52,74
tai100a	21052466	**0,765(0)**	1349,022	0,846	145,26	0,848	129,73	0,825	142,06
tai20b	122455319	**0,000(10)**	0,178	0,000	0,23	0,000	0,23	0,000	0,23
tai25b	344355646	**0,000(10)**	0,606	0,000	0,46	0,000	0,46	0,000	0,56
tai30b	637117113	**0,000(10)**	3,440	0,000	1,28	0,000	1,30	0,000	1,31
tai35b	283315445	**0,000(10)**	1,999	0,000	2,44	0,000	2,44	0,000	2,39
tai40b	637250948	**0,000(10)**	1,886	0,000	3,17	0,000	3,18	0,000	3,18
tai50b	458821517	**0,000(10)**	28,758	0,000	9,54	0,000	8,63	0,000	8,82
tai60b	608215054	**0,000(10)**	26,662	0,000	18,56	0,000	19,48	0,000	17,08
tai80b	818415043	0,008(0)	827,93	0,013	49,02	**0,006**	57,29	**0,006**	58,24
tai100b	1185996137	0,203(0)	1124,549	**0,040**	107,63	0,044	123,15	0,056	118,91
sko42	15812	**0,000(10)**	2,734	0,000	3,86	0,000	3,74	0,000	3,98
sko49	23386	**0,000(10)**	19,154	0,004	9,60	0,014	8,52	0,008	9,61
sko56	34458	**0,000(10)**	12,767	0,002	13,75	0,002	14,22	0,002	13,16
sko64	48498	**0,000(10)**	6,729	0,000	25,44	0,000	24,00	0,000	22,03
sko72	66256	**0,000(10)**	37,250	0,014	35,29	0,013	34,50	0,006	37,98
sko81	90998	**0,003(9)**	99,407	0,017	62,36	0,023	56,25	0,016	56,36
sko90	115534	**0,012(0)**	1028,629	0,017	99,69	0,013	92,08	0,026	89,60
sko100a	152002	0,036(1)	1234,106	0,026	134,53	**0,024**	132,80	0,027	129,22
sko100b	153890	0,011(0)	1298,457	0,011	124,84	0,010	113,02	**0,008**	106,55
sko100c	147862	**0,001(0)**	1290,869	0,008	113,95	0,010	107,90	0,006	126,69
sko100d	149576	0,015(3)	1028,149	0,016	129,23	**0,011**	121,98	0,027	123,45
sko100e	149150	0,014(0)	1296,826	**0,007**	130,14	0,011	115,62	0,009	108,84
sko100f	149036	0,016(0)	1295,274	0,021	118,90	**0,012**	127,00	0,023	110,28
Moyenne totale		**0,103(173)**	445,700	0,120(150)	46,93	0,119(151)	44,96	0,112(154)	44,86
Moyenne type 2		**0,318(40)**	350,038	0,391	29,00	0,390	25,12	0,360	26,94
Moyenne type 3		0,023(70)	224,001	**0,006**	21,37	**0,006**	24,02	0,007	23,40
Moyenne type 4		**0,008(63)**	665,412	0,011	77,03	0,011	73,20	0,012	72,13

TABLE 3.11 – Accélération GPU

Instances(5)	**BKS**	**D-HITS**				
		déviation	$temps_{CPU}$(s)	$temps_{GPU}$(s)	ACC	Gain en heures
wil100	273038	0,006(0)	**807,649**	1339,718	0,6	-0,15
tho150	8133398	0,065(0)	**2873,837**	3459,79	0,8	-0,16
tai150b	498896643	1,250(0)	**2861,015**	3376,469	0,8	-0,14
tai256c	44856925	0,218(0)	17801,394	**10346,5**	1,7	2,07
tai343e01	136288	22,401(0)	41240,9	**32693**	1,3	2,37

Dans le tableau 3.11, on observe que $D-HITS_{CPU}$ est plus rapide que $D-HITS_{GPU}$ jusqu'à la taille 150 car le nombre de threads lancés reste petit et ne peut pas compenser le temps perdu dans les transferts. Avec l'instance de taille 256, nous obtenons la meilleure accélération de $\times$1,7. Dans le cas de l'instance de taille 343, l'accélération rechute à $\times$1,3 mais la version GPU reste plus rapide que la version CPU avec un gain de 2,37 heures en temps de calcul. Cette rechute s'explique par la saturation des mémoires du GPU car on génère 343^2 threads qui utilisent beaucoup de mémoire. Néanmoins, nous avons une version efficace telle que le tableau 3.10 le montre, et une implémentation plus rapide à partir des instances de taille 256 comme le montre le tableau 3.11.

La parallélisation au niveau des structures de voisinage sur le GPU a été additionnée à nos trois meilleures variantes distribuées. Cette parallélisation a pour but de résoudre des problèmes de taille 343 qui sont rarement traités dans la littérature. Les variantes utilisées sont le DISCO-HITS-DG, le DISCO-HITS-UX et le DM-HITS.

La suite des expérimentations est dédiée exclusivement aux instances de taille 343, proposées par Drezner et al. (2005) (*http ://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html*). A notre connaissance, seuls les travaux de Drezner et al. (2005) et Hussin & Stützle (2009) ont tentés de résoudre des instances de cette taille. Dans ces deux travaux, seule l'instance tai343e01 est traitée. Les instances de taille 343 sont considérées comme des instances très difficiles à résoudre pour les métaheuristiques. Ces instances sont d'un nouveau type. Elles sont structurées et symétriques. Un des premiers résultats qui utilise

les métaheuristiques est donné par Drezner et al. (2005), la déviation par rapport au BKS est de 18%. Dans le travail de Hussin & Stützle (2009), quelques propositions sont comparées pour cette instance. Seul le meilleur résultat parmi 20 tentatives est donné dans leur travail. Les algorithmes utilisés dans leur comparaison sont : le Ro-TS, les variantes 2 et 3 de leur algorithme nommé HILS, et le ILS-ES de Stützle (2006). Le meilleur résultat est obtenu par HILS(3) avec 0%. HILS(2) obtient 59%, ILS-ES obtient 4% et Ro-TS obtient 34%.

Le tableau 3.12 propose les résultats obtenus à l'aide de nos trois variantes distribuées et parallélisées sur GPU. Toutes les variantes utilisent l'évaluation sur le GPU. Pour l'instance tai343e01, le $DM-HITS_{GPU}$, qui est la meilleure variante, obtient une moyenne de 24% et son meilleur résultat est de 21% par rapport au BKS. Notre proposition surpasse 2 algorithmes parmi les 5 expérimentés dans les travaux de Drezner et al. (2005) et Hussin & Stützle (2009) pour la meilleure tentative. Ces auteurs n'ayant fourni de résultats que pour une seule instance de taille 343, cela ne nous permet pas de tirer des conclusions pour ce type d'instances. Notre objectif principal est d'explorer plus d'instances de taille 343 et de produire de nouveaux résultats en traitant les instances proposées dans la littérature.

TABLE 3.12 – Comparaison entre les $D-HITS_{GPU}$

Instances(10)	**BKS**	**constraintes**	$DISCO-HITS-DG_{GPU}$		$DISCO-HITS-UX_{GPU}$		$DM-HITS_{GPU}$	
			valeurs	temps	valeurs	temps	valeurs	temps
tai343e01	136288	61202	169837,4	6006,625	189081	5999,061	**169469**	6004,44
tai343e02	-	61294	177418	6061,415	199644,2	6048,365	**176513,8**	6050,463
tai343e03	-	61052	169801,2	6046,138	174085	6051,205	**169337**	6049,731
tai343e04	-	61370	183178,6	6033,128	185610,6	6044,743	**181480,8**	6031,543
tai343e05	-	61170	**170440,2**	5988,235	172534,8	5999,924	186466,4	5994,56
tai343e06	-	61216	171274	6008,4	192321,4	6003,938	**169404,4**	5981,742
tai343e07	-	61250	173455,2	6050,997	177665	6051,386	**173350,4**	6028,06
tai343e08	-	61122	161009,6	6014,866	183656	6012,184	**157866,2**	5988,639
tai343e09	-	61226	166429,8	6056,832	169797,2	6049,528	**165516,6**	6037,449
tai343e10	-	61246	176980,2	5989,135	180842	5992,674	**176851,2**	5970,777

La première comparaison sur le tableau 3.12 est entre $DISCO-HITS-DG_{GPU}$ et $DISCO-HITS-UX_{GPU}$. Le $DISCO-HITS-DG_{GPU}$ obtient de meilleurs résultats sur toutes les 10 instances expérimentées. La deuxième comparaison est entre $DISCO-HITS-DG_{GPU}$ et $DM-HITS_{GPU}$: l'algorithme $DM-HITS_{GPU}$ arrive à battre le $DISCO-HITS-DG_{GPU}$ sur 9 instances parmi les 10. Ce tableau nous permet de conclure que notre proposition $DM-HITS_{GPU}$ est celle qui est la plus efficace sur les instances de taille 343. Pour cette raison, seule cette proposition sera considérée dans le reste de ce chapitre.

Dans le tableau 3.13, on compare le $DM-HITS_{GPU}$ à l'un des algorithmes les plus populaires et les plus utilisés dans la littérature qui est le Ro-TS. Puisque nous ne connaissons ni le BKS ni la valeur optimale pour 9 instances parmi les 10 testées dans le tableau 3.13, les valeurs obtenues par le Ro-TS représentent notre BKS appelé BKSROTS dans cette comparaison. La déviation (Dv) est calculée avec l'équation 3.3 :

$$Dv = \frac{(solution - BKSROTS) \times 100}{BKSROTS} \tag{3.3}$$

Le nombre de contraintes est calculé pour chaque instance afin de donner une idée de la complexité des instances. Le temps est calculé en secondes. La Dv est calculée par l'équation 3.3 et l'ACC représente l'accélération entre notre proposition distribuée, massivement parallèle, et la version séquentielle du Ro-TS. Elle est calculée en divisant le temps que le Ro-TS a réalisé par le temps que le $DM-HITS_{GPU}$ a réalisé. Le même nombre d'appels de la fonction objectif est utilisé dans les deux algorithmes, soit $60000 \times n$ pour le Ro-TS et $6 \times 1000 \times n \times 10$ pour le $DM-HITS_{GPU}$, i.e. 6 processus qui lancent 10 RT où chaque RT exécute $1000 \times n$ itérations (voir tableau 3.9).

TABLE 3.13 – Comparaison avec Ro-TS pour les instances de taille 343

Instances(10)	**BKS**	**constraintes**	Ro-TS		$DM-HITS_{GPU}$			
			valeur	$temps_{CPU}$	valeur	$temps_{CPU-GPU}$(s)	Dv (%)	ACC
tai343e01	136288	61202	190228,2	49493,383	**169469**	6004,44	-10,9	8,2
tai343e02	-	61294	**162733,8**	48383,697	176513,8	6050,463	8,5	8
tai343e03	-	61052	**152729,2**	48499,189	169337	6049,731	10,9	8
tai343e04	-	61370	**171786,2**	48366,099	181480,8	6031,543	5,6	8
tai343e05	-	61170	**153593,4**	47313,967	186466,4	5994,56	21,4	7,9
tai343e06	-	61216	171394,4	48319,116	**169404,4**	5981,742	-1,2	8,1
tai343e07	-	61250	197057	49439,529	**173350,4**	6028,06	-12	8.2
tai343e08	-	61122	165106,8	48352,558	**157866,2**	5988,639	-4.4	8.1
tai343e09	-	61226	**152292**	49707,625	165516,6	6037,449	8,7	8,2
tai343e10	-	61246	**161535,8**	48375,878	176851,2	5970,777	9,5	8,1

Le nombre de contraintes varie entre 61052 et 61370. Le $DM-HITS_{GPU}$ surpasse le Ro-TS sur 4 instances parmi 10 en qualité de solution. Le temps de calcul est réduit grâce à l'évaluation parallèle des voisins sur GPU et la distribution des évaluations entre les processus. Cela permet à notre approche d'obtenir une accélération moyenne de ×8,08. Au niveau de la qualité des résultats, notre proposition surpasse le Ro-TS pour 4 instances avec une déviation entre -12% et -1,2%. En considérant l'accélération moyenne de ×8 et la qualité des solutions obtenues, notre proposition est compétitive par rapport au Ro-TS.

3.4 Discussion

Le but principal à travers l'étude du QAP est de proposer plusieurs méthodes distribuées et massivement parallèles qui utilisent quelques uns des niveaux d'hybridation et de parallélisation déjà présentés dans le deuxième chapitre. L'objectif principal étant la qualité des solutions, plusieurs résultats compétitifs par rapport aux meilleurs travaux de la littérature sont proposés dans ce chapitre.

À travers l'étude de ce problème, nous avons pu analyser plusieurs méthodes de diversification et d'échange de données sur la parallélisation au niveau des algorithmes. Nous

avons pu également confirmer les hypothèses que nous avons émises sur l'amélioration des résultats avec l'échange de données et sur l'efficacité de l'hybridation.

Dans la suite de ce travail, grâce aux différentes expérimentations menées sur les niveaux d'hybridation et de parallélisation, nous avons pu nous focaliser sur un problème plus complexe et peu commun dans l'optimisation combinatoire qui est le problème de la résolution structurale des zéolithes. La résolution de ce problème est très utile aux chercheurs du laboratoire de chimie de Mulhouse. L'objectif est de proposer de nouvelles formulations ainsi que des métaheuristiques efficaces capables de générer des structures stables énergétiquement.

Conclusion

Dans ce chapitre, plusieurs algorithmes distribuées et massivement parallèles ont été proposés. Différentes techniques de diversification et d'échange de données ont été présentées et comparées. Plusieurs expérimentations ont été réalisées pour valider les algorithmes proposés.

Dans le chapitre suivant, le problème de la résolution structurale des zéolithes est présenté, ainsi que les différentes méthodes élaborées pour le résoudre.

Chapitre 4

Le problème de la résolution structurale des zéolithes

Dans ce dernier chapitre, nous traitons le problème de la résolution structurale des zéolithes. Dans la première partie, nous présentons la problématique ainsi que deux formulations de la fonction objectif. Par la suite, nous présentons deux versions d'un algorithme génétique qui utilisent l'hybridation à bas niveau et le niveau de la parallélisation des individus. La première simulation est exécutée sur la première formulation avec les deux versions et appliquée sur trois benchmarks de zéolithe cible. La deuxième simulation est menée avec la deuxième version à laquelle on ajoute une parallélisation au niveau des algorithmes et une hybridation à haut niveau. Cette version est expérimentée sur six benchmarks de zéolithe cible. Les deux versions montrent une capacité à trouver la valeur optimale de la fonction objectif. L'analyse chimique de plusieurs topologies générées a validé des topologies stables énergétiquement et même de nouvelles structures zéolitiques non connues.

4.1 Présentation du problème de la résolution structurale des zéolithes

4.1.1 Description

Les zéolithes sont des minéraux microporeux dont le diamètre des pores, selon la nomenclature *IUPAC* (McCusker et al., 2001), est inférieur à 20 Å. Leurs structures résultent de l'assemblage de tétraèdres TO_4 (avec T = Silicium (Si) ou Aluminium (Al)). Chaque atome d'oxygène (O) est commun à deux tétraèdres (Figure 4.1). La microporosité est ordonnée et régulière, conduisant à des canaux et des cages réparties régulièrement dans l'espace (Figure 4.2). La taille des pores est de l'ordre de celles des molécules conventionnelles et elle est à l'origine du terme "tamis moléculaire".

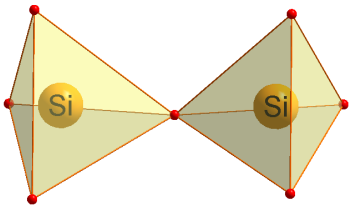


FIGURE 4.1 – Deux tétraèdres SiO_4 liés par l'intermédiaire d'un atome d'oxygène.

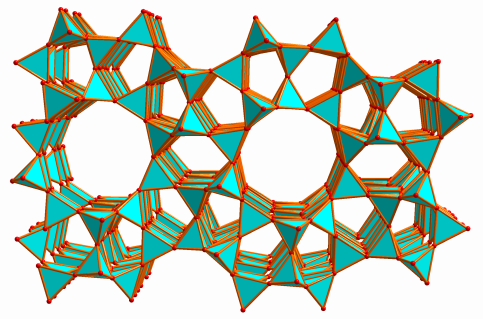


FIGURE 4.2 – La structure de la zéolithe ZSM-5 (Kokotailo et al., 1978) avec ses pores moyens délimités par 10 tétraèdres.

En théorie, il existe un nombre infini de structures zéolitiques possibles (Delgado et al., 1999; Treacy et al., 2004), mais seules 231 structures zéolitiques sont aujourd'hui reconnues par la Commission SC-IZA (de l'anglais *Structure Commission of the International Zeolite Association*) (Baerlocher & McCusker). Parmis les structures reconnues, seules 40 zéolithes sont naturelles. Le système de pores d'une zéolithe peut être mono, bi ou tridimensionnel, et chaque système est désigné par sa topologie. Les ouvertures des canaux sont généralement caractérisées par le nombre d'éléments constitutifs d'atomes T que nous appelons MR (de l'anglais *Membered Ring*) (*e.g.*, 10MR signifie 10 tétraèdres en anneaux qui forment l'entrée d'un canal, comme dans le cas de ZSM-5 dans la figure 4.2). Lorsqu'une nouvelle topologie est découverte, un code à trois lettres est attribué par le SC-IZA. Ce code à trois lettres est généralement lié au nom de la matière ou de l'équipe responsable de la découverte de cette topologie. Par exemple, le code de la topologie **MFI** a été attribué à la zéolithe ZSM-5 (de l'anglais *Zeolite Socony **M**obil-5 (**FI**ve)*) (Argauer & Landolt, 1972) et la topologie **UTL** pour la zéolithe IM-12 (IFP Mulho**U**se-**T**w**L**ve) (Paillaud et al., 2004). Comme tout matériau cristallin, les zéolithes obéissent aux lois de la cristallographie. En bref, dans une structure cristalline, il est possible de définir un objet à répéter par une simple translation selon une direction particulière, cette répétition permet de former un réseau cristallin. Cet objet, qui est la plus petite unité du réseau cristallin est appelé UC (*Unité Cellulaire* ou en anglais *Unit Cell*) (Figure 4.3).

L'UC complète est générée à partir de l'unité asymétrique qu'on nomme AU (de l'anglais *Asymmetric Unit*) après l'application des opérateurs de symétrie d'un groupe d'espace donné. Cela signifie que nous pouvons également classer les structures par la symétrie qu'elles possèdent, donc, le groupe d'espace auquel elles appartiennent. En raison des propriétés de l'espace géométrique, il existe seulement 230 groupes d'espace en cristallographie. Le groupe d'espace est désigné soit par son numéro (1-230), soit par des symboles qui décrivent les symétries de ce groupe : une lettre majuscule suivie d'une succession de chiffres et de lettres (exemple, pour ZSM-5, le groupe d'espace est *Pnma*, #62).

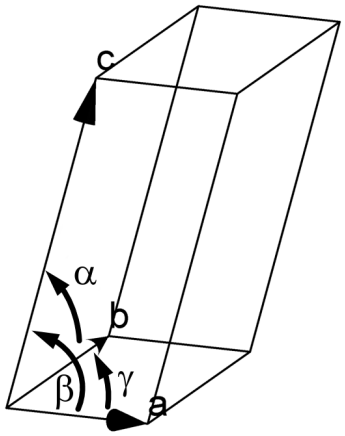


FIGURE 4.3 – Les 6 paramètres définissant une UC, a, b et c sont en Å, et a, β et γ en °.

La figure 4.4 représente une UC avec l'AU avant et après l'application des opérateurs de symétrie, respectivement.

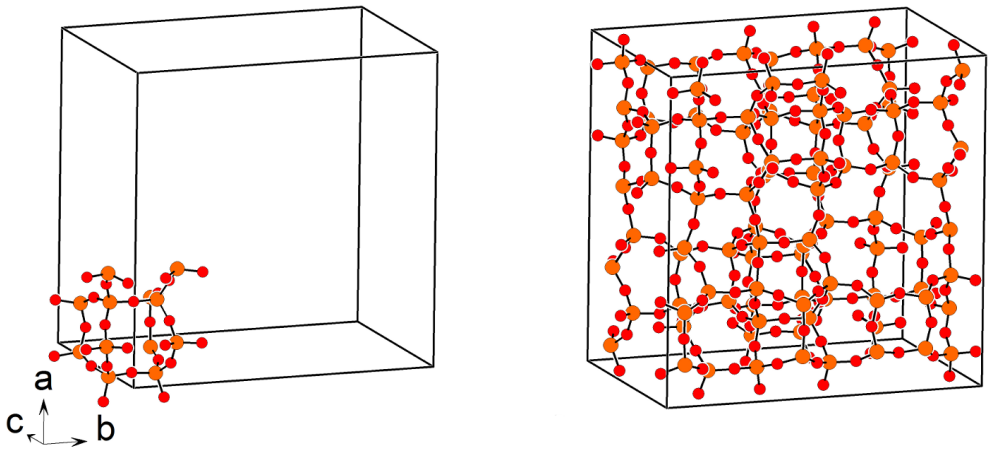


FIGURE 4.4 – Une UC de la zéolithe ZSM-5 avec la topologie **MFI**, avec l'AU avant (gauche) et après l'application des opérateurs de symétrie (droite). Les disques orangés et rouges représentent des sphères de silicium et des atomes d'oxygène, respectivement.

4.1.2 Positionnement

Depuis la découverte de la première zéolithe naturelle par le minéralogiste suédois Axel Fredrik Cronstedt (Cronstedt, 1756) jusqu'aux applications industrielles les plus récentes, les zéolithes comptent 250 ans d'histoire. Au cours des deux siècles qui ont suivi cette première découverte, la science des zéolithes a été limitée aux zéolithes naturelles découvertes dans les grands bassins sédimentaires. Il est à noter qu'il est presque impossible d'avoir des lots de zéolithes naturelles qui sont homogènes dans la composition et les propriétés. Cette contrainte limite considérablement leur utilisation industrielle et les rend réservées à des fins spéciales directement liées à l'homme et à son environnement (l'isolation des matériaux de construction, la pisciculture, l'élevage, etc.). Ce n'est qu'à partir des années 1930, et des travaux sur les propriétés de synthèse et d'adsorption, que la science des zéolithes a vraiment décollé (Barrer, 1982; Guisnet & Gilson, 2002). Au cours des années 1950, les premiers procédés pour la synthèse des zéolithes A, X et Y sont développés avant leur commercialisation par la société *Union Carbide* (Barrer, 1982).

L'un des défis de la recherche dans le domaine des zéolithes est la découverte de nouveaux matériaux microporeux de type zéolithe. Ils sont utilisés pour des applications en catalyse, séparation et adsorption. Les zéolithes sont particulièrement intéressantes pour l'industrie pétrolière pour avoir de nouveaux catalyseurs zéolitiques à larges pores pour favoriser les phénomènes de diffusion et/ou de faire de nouvelles réactions avec des molécules plus grandes en taille. Les zéolithes sont composées principalement d'atomes de silicium et d'aluminium qui représentent deux des éléments les plus abondants sur terre. D'un point de vue expérimental, la synthèse d'une nouvelle topologie de zéolithe reste toujours le plus grand défi et la partie la plus difficile. Pour cette raison, d'une manière générale, la découverte d'une nouvelle zéolithe est plus ou moins fortuite (Li et al., 2015) et elle dépend d'un processus d'essais et d'erreurs. Pour synthétiser une nouvelle zéolithe, dans la grande majorité des cas, un agent d'orientation de la structure organique nommé OSDA (de l'anglais *Organic Structure Directing Agent*) est nécessaire. En termes

de conception d'OSDA, les exemples de l'utilisation de la modélisation moléculaire sont rares, la zéolite ZSM-18 étant le plus célèbre exemple (Schmitt & Kennedy, 1994). Cependant, une méthode plus élaborée proposée par Lewis *et al.* prédit un OSDA efficace qui a conduit à un tamis moléculaire ayant la topologie de la *chabazite* dans le système aluminophosphate (Lewis et al., 1997; Speybroeck et al., 2015). Une méthodologie similaire a été développée par Pophale et al. (2013) et Schmidt et al. (2014). Les deux concepts sont intéressants, mais ont besoin d'autres preuves pour être considérée comme véritablement efficaces.

Les AG ont la capacité d'explorer l'espace de recherche et de trouver des solutions inattendues. Ils ont été utilisés avec succès pour résoudre des problèmes de la chimie combinatoire tels que le problème de *Lennard Jones Cluster* (LJC) (Eshelman & Schaffer, 1993; Bäck, 1996b). Pour un groupe donné de N atomes, le problème de LJC consiste à trouver les positions relatives des atomes dans un espace euclidien à trois dimensions afin de minimiser l'énergie potentielle de la structure (Fan, 2002). Différents travaux ont également utilisé les AG pour résoudre le problème de LJC tels que ceux publiés par Daven et al. (1996) et Barrón et al. (1999). Cette métaheuristique a montré une bonne capacité à résoudre les problèmes de chimie (Archibald et al., 2005).

Le problème de la résolution structurale des zéolithes ou ZSP (de l'anglais *Zeolites Structure Problem*), introduit par Falcioni & Deem (1999), est similaire aux concepts du problème de LJC. En effet, le ZSP peut être classé dans le domaine de la chimie combinatoire et comme un problème NP-difficile proche du problème de LJC. Baumes *et al.* ont développé un AG intégré dans la bibliothèque EASEA (Collet et al., 2000) et parallélisé sur GPU (Baumes et al., 2011, 2013). L'accélération avec GPU était l'objectif principal de leurs travaux. Nous pouvons également mentionner deux programmes informatiques spécifiques indépendants, qui sont liés au ZSP, à savoir *FOCUS* (Grosse-Kunstleve et al., 1997, 1999) et *Zefsa II* (Deem & Newsam, 1989; Deem et al., 1992; Falcioni & Deem, 1999). Ce

sont des méthodes spécifiques pour les solutions de structure zéolitique générées à partir des données de diffraction de poudre.

Contrairement aux deux programmes *FOCUS* et *Zefsa II* qui utilisent le diagramme de poudre, nous proposons un algorithme qui est basé sur l'évaluation avec des pénalisations et des récompenses des structures pour converger vers une topologie viable. Cette évaluation donne une liberté à l'algorithme pour lui permettre de trouver de nouvelles topologies non répertoriées.

4.1.3 Formulations de la fonction objectif

4.1.3.1 Formulation 1

Dans ce chapitre, nous proposons une première formulation de la fonction objectif pour résoudre le ZSP. Le calcul est réalisé sur la totalité de l'UC pour évaluer le potentiel de la structure à être une topologie d'une zéolithe stable. Avant chaque évaluation, l'UC de la solution est générée à partir des positions aléatoires des atomes initiaux et des paramètres d'entrée. L'UC est évaluée à l'aide de notre fonction objectif. Cette fonction objectif regroupe un ensemble de pénalisations (contraintes) destinées à évaluer le potentiel de la structure zéolitique. Le ZSP peut être vu comme un problème de minimisation, ce qui veut dire que si toutes les contraintes sont satisfaites alors nous obtenons une structure avec la valeur optimale qui est égale à zéro dans la formulation 1. Cette structure est alors considérée comme une topologie optimale d'une zéolithe potentielle.

Cette formulation de la fonction objectif est composée de 5 pénalisations. Chaque pénalisation est calculée indépendamment et ajoutée à l'évaluation finale. Les équations de 4.1 à 4.6 proposent une première formulation de la fonction objectif (formulation 1).

$$Min \sum_{i=1}^{5} C_i \tag{4.1}$$

$$C_1 = Nb_{structureconnectée} - 1 \tag{4.2}$$

$$C_2 = \sum_{j=1}^{n} |NbLiens_j - 4| \tag{4.3}$$

$$C_3 = \sum_{j=1}^{n} P : \begin{cases} P = 1, & S'il \quad existe \quad un \quad 3MR_j \\ P = 0, & Sinon \end{cases} \tag{4.4}$$

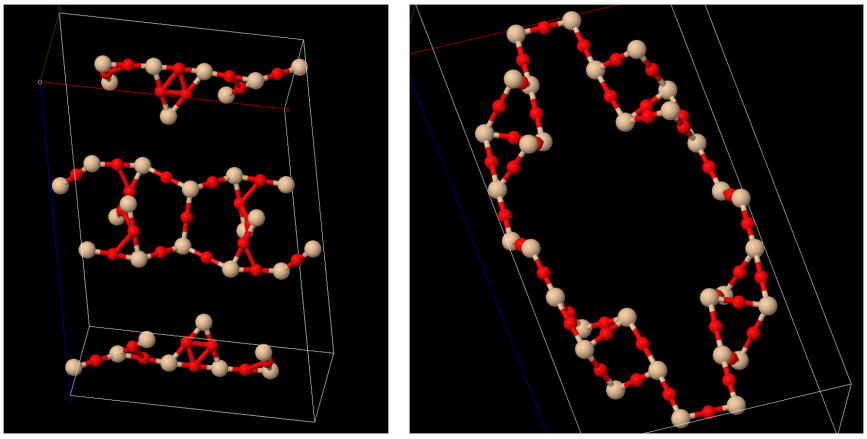
$$C_4 = \sum_{j=1}^{n} P : \begin{cases} P = 1, & Si \quad DisA_j <= L \\ P = 0, & Sinon \end{cases} \tag{4.5}$$

$$C_5 = \sum_{j=1}^{n} P : \begin{cases} P = 1, & Si \quad An_j <= L1 \quad ou \quad An_j >= L2 \\ P = 0, & Sinon \end{cases} \tag{4.6}$$

La première pénalisation (C_1) traite la connexion entre tous les atomes T de l'UC. Tous les atomes T générés doivent être connectés les uns aux autres par un atome O pour créer une zéolithe viable. La valeur de $Nb_{Structureconnectée}$ (voir l'équation 4.2) est égale à 1 si l'UC est composée d'une seule structure. La figure 4.5 montre la bonne connectivité avec une seule structure connectée et la mauvaise connectivité avec différentes structures non connectées. La structure avec une bonne connectivité aura une valeur nulle dans sa pénalisation C_1.

La seconde pénalisation (C_2) est la formation des tétraèdres. Chaque atome T a besoin de 4 liens avec les autres atomes T pour devenir stable. L'algorithme évalue tous les n atomes T (nombre d'atomes T dans l'UC) et calcule le nombre de liens pour chaque atome (NbLiens dans l'équation 4.3). Si cette valeur n'est pas égale à 4, la structure est pénalisée.

La troisième pénalisation (C_3) vérifie l'existence des 3MR. Un 3MR se compose de trois tétraèdres successifs reliés entre eux. Cette forme des 3MR rattachés est impossible à synthétiser dans les systèmes de silice ou aluminosilicate purs. Ces 3MR se trouvent par



(a) (b)

FIGURE 4.5 – Deux exemples de connectivités, (a) mauvaise connectivité, (b) bonne connectivité.

contre en silice à base de matériaux microporeux qui contiennent des éléments hétérogènes tels que le béryllium, le germanium ou le zinc (Baerlocher & McCusker). Puisque nous orientons notre travail vers les systèmes purs, à chaque fois qu'un 3MR est détecté, la structure est pénalisée.

La quatrième pénalisation (C_4) a pour but de vérifier les distances entre les atomes. Deux atomes T doivent être à la bonne distance pour qu'ils soient considérés comme connectés par l'algorithme. Dans notre formulation, cette distance doit être ni trop loins, pour que les atomes T soit connectés par un atome O, ni trop proche, pour les empêcher d'être collés. En règle générale, pour les zéolithes de silice pure, la distance Si-Si entre deux atomes de silicium voisins est d'environ 3,1 Å. Dans toute la formulation, notre algorithme considère que deux atomes sont connectés si et seulement si leurs distances se situe entre 2,8 Å et 3,3 Å. La pénalisation C_4 (voir l'équation 4.5) propose une pénalisation supplémentaire sur cette distance. Elle parcourt tous les atomes T et vérifie si la distance entre deux atomes T n'est pas inférieure à la valeur limite L. Cette valeur est fixée à 2,44 Å, ce qui signifie que chaque distance inférieure à 2,44 Å est pénalisée.

La pénalisation finale (C_5) de la formulation 1 traite les angles $\widehat{TTT}$ entre les atomes T. La contrainte C_5 permet de pénaliser les angles formés par T s'ils n'appartiennent pas à une plage fixe (*L1* et *L2*, voir l'équation 4.6, où *L1* et *L2* sont respectivement de 60 ° et 180 °).

Les pénalisations 4.5 et 4.6 définissent la stabilité énergétique de la structure. La figure 4.6 présente la formation d'angles entre le centre de deux tétraèdres d'une structure de zéolithe. L'algorithme 16 présente le pseudo-code de l'évaluation des angles.

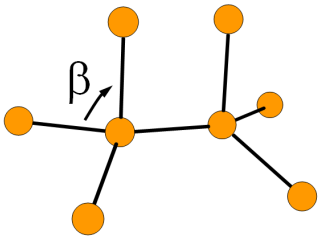


FIGURE 4.6 – Les angles β entre les atomes de silicium $\widehat{TTT}$.

Algorithme 16 L'évaluation des angles

Entrées : *NbA* : le nombre total d'atomes T dans UC ; *An* : la valeur de l'angle trouvée au niveau d'un atome T
Sortie : C_5 : la valeur de la pénalisation 5
An := 0 ;
pour *i* := 1 à *NbA-2* **faire**
    pour *j* := *i*+1 à *NbA-1* **faire**
        si *atome(i)* a un lien avec *atome(j)* **alors**
            pour *k* := *j*+1 à *NbA* **faire**
                si *atome(k)* a un lien avec *atome(j)* **alors**
                    An := calcul de la valeur de l'angle $\widehat{ijk}$ avec le théorème d'Al-Kashi (Honvault, 2004) ;
                fin si
            fin pour
        fin si
    fin pour
fin pour
si *An* n'est pas dans l'intervalle délimité par *L1* et *L2* **alors**
    C_5++ ;
fin si

4.1.3.2 Formulation 2

La deuxième formulation proposée dans ce chapitre reprend les mêmes contraintes que la formulation 1 auxquelles on ajoute deux récompenses. Le point faible de la première formulation est que, lorsque nous avons une solution qui satisfait aussi toutes les contraintes, rien ne garantit qu'il n'y a pas une autre solution qui satisfait toutes les conditions et qui est meilleure que la solution sélectionnée. L'idée de cette formulation est d'exécuter le programme avec la formulation 1 jusqu'à atteindre la valeur 0. Toutes les structures qui ont atteint la valeur zéro sont réévaluées avec les deux équations de récompense. La valeur minimale, qui peut être négative dans ce cas, est retenue par l'algorithme. Les équations 4.7 et 4.8 proposent les récompenses ajoutées à la formulation 1 de la fonction objectif pour créer la formulation 2.

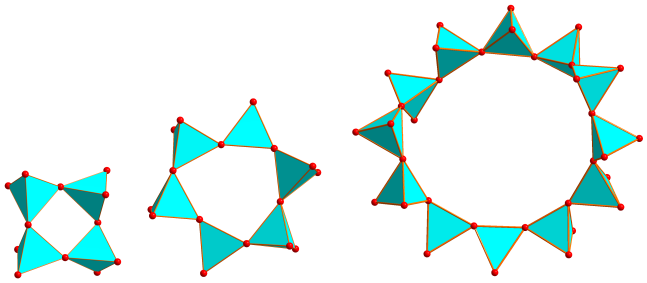
$$R_1 = \sum_{j=1}^{n} P : \begin{cases} P = -X, & Si \quad XMR_j \\ P = 0, & Sinon \end{cases} \tag{4.7}$$

$$R_2 = \sum_{j=1}^{n} P : \begin{cases} P = -1, & S'il \quad existe \quad un \quad 4MR_j \\ P = 0, & Sinon \end{cases} \tag{4.8}$$

La première récompense de la formulation 2 (R_1) concerne la création des XMR. Les zéolithes sont caractérisées par la formation de ces XMR qui permettent par la suite la formation des pores. Ils sont à l'origine de la porosité des zéolithes et donc de leur capacité d'adsorption des molécules. Ces pores doivent être larges pour devenir intéressant. l'algorithme calcule les XMR jusqu'à 12-MR et il récompense la structure par la valeur X de l'XMR trouvée.

La deuxième récompense de cette formulation (R_2) traite la création des 4MR dans les structures. Ces 4MR sont très intéressantes pour les structures zéolitiques. C'est un élément constitutif de plusieurs zéolithes. L'algorithme calcule le nombre des 4MR et il récompense la structure avec ce nombre.

L'objectif de ces deux récompenses est de minimiser la valeur des structures les plus intéressantes. La figure 4.7 présente quelques uns des XMR les plus communs et les plus rencontrés dans les structures zéolitiques.



(a) (b) (c)

FIGURE 4.7 – Des exemples de trois anneaux différents présents dans les zéolithes, (a) 4MR, (b) 6MR et (c) 12MR.

4.2 Présentation des méthodes

4.2.1 L'algorithme P-GHAZ

La première proposition pour résoudre le problème de la résolution structurale des zéolithes est un AG hybride et parallèle que nous avons nommé P-GHAZ (de l'anglais *Parallel Genetic Hybrid Algorithm for Zeolite*). Le P-GHAZ utilise **le niveau de parallélisation des individus** et **l'hybridation à bas niveau**. L'algorithme 17 présente le pseudo-code de cette approche.

Chaque position de l'atome dans l'AU est définie par 3 coordonnées atomiques (x, y, z). Pour créer une zéolithe potentielle par le P-GHAZ, l'algorithme modifie les coordonnées des atomes dans l'AU. En utilisant les paramètres de l'UC, la topologie est créée et évaluée. L'UC est l'unité de volume la plus petite qui contient toutes les informations structurelles et les informations de symétrie.

Algorithme 17 Le pseudo-code de P-GHAZ

1: *Entrées* : *initAtoms* : le nombre total des atomes initiaux dans l'unité asymétrique ; N : la taille de la population ; *step* : la valeur du pas qui est effectué par l'atome dans la mutation ; *ObjF* : la fonction objectif
2: *Sorties* : *currentFitness* : la fitness de l'individu courant ; *atome* : les positions des atomes T d'un individu de la population
3: /* Initialisation */
4: **pour tous** les N individus de la population en parallèle **faire**
5: **pour** $i := 1$ à *initAtoms* **faire**
6: Générer une position aléatoire **x** de *atome(i)* à l'intérieur des bornes de l'UC ;
7: Générer une position aléatoire **y** de *atome(i)* à l'intérieur des bornes de l'UC ;
8: Générer une position aléatoire **z** de *atome(i)* à l'intérieur des bornes de l'UC ;
9: **fin pour**
10: **fin pour**
11: **répéter**
12: /* Sélection */
13: **pour tous** les N individus de la population en parallèle **faire**
14: Sélectionner aléatoirement un parent de la population ;
15: **fin pour**
16: /* Croisement */
17: **pour tous** les N individus de la population en parallèle **faire**
18: Croisement entre l'individu courant et le parent sélectionné ;
19: **fin pour**
20: /* Mutation */
21: Sélectionner aléatoirement un pourcentage prédéfini d'individus mutants ;
22: **pour tous** les individus mutants en parallèle **faire**
23: Sélectionner aléatoirement un atome mutant de l'individu mutant ;
24: RL de l'atome mutant selon la valeur *step* (voir algorithme 18) ;
25: **fin pour**
26: /* Evaluation */
27: **pour tous** les N individus de la population en parallèle **faire**
28: Evaluer la *currentFitness* de l'individu avec *ObjF* ;
29: **fin pour**
30: **jusqu'à** ce que les conditions d'arrêt soient satisfaites

L'objectif est de générer de nouvelles structures avec des topologies qui sont énergétiquement stables et potentiellement synthétisables en zéolithes lorsque la topologie est inconnue. Ces structures sont obtenues à travers des paramètres d'entrée déterminés par l'utilisateur qui sont : le nombre d'atomes T dans l'AU, les dimensions de l'UC et le groupe d'espace. Cette stratégie peut être utilisée pour découvrir des topologies connues ou inconnues.

Le but de notre proposition est de trouver les positions des atomes permettant de générer de potentielles topologies stables et inconnues. Les algorithmes évolutionnaires sont naturellement adaptés à proposer de nouvelles structures (Jones et al., 1995). Notre approche P-GHAZ utilise la bibliothèque EASEA qui gère le parallélisme des individus sur le GPU.

4.2.1.1 Représentation et sélection

Notre algorithme génère une population aléatoire. Chaque individu de cette population représente l'AU, ce qui signifie qu'il représente les coordonnées initiales des atomes (x, y, z). Par exemple, si l'AU d'une zéolithe est composée de 4 atomes T, l'individu est composé de 12 coordonnées. L'opérateur de sélection utilise un tournoi entre l'individu courant et un autre individu aléatoire. L'algorithme P-GHAZ gère uniquement les positions des atomes T. Les atomes d'oxygène sont ajoutés à la fin de l'algorithme au centre des deux atomes T susceptibles d'être reliés.

4.2.1.2 Opérateurs de croisement

Pour l'algorithme P-GHAZ, nous avons mis en place deux stratégies de croisement. La première stratégie est le croisement à un seul point OPX (de l'anglais *One Point Crossover*). Il effectue un croisement entre les atomes de deux parents, à une position aléatoire.
Exemple du OPX :

Croisement(parent1 ;parent2 ;position 3)

Parent1($\underline{\mathbf{x_{1.1}}, \mathbf{y_{1.1}}, \mathbf{z_{1.1}}}$; $\underline{\mathbf{x_{1.2}}, \mathbf{y_{1.2}}, \mathbf{z_{1.2}}}$; $\underline{\mathbf{x_{1.3}}, \mathbf{y_{1.3}}, \mathbf{z_{1.3}}}$; $x_{1.4}, y_{1.4}, z_{1.4}$)

Parent2($x_{2.1}, y_{2.1}, z_{2.1}$; $x_{2.2}, y_{2.2}, z_{2.2}$; $x_{2.3}, y_{2.3}, z_{2.3}$; $\underline{\mathbf{x_{2.4}}, \mathbf{y_{2.4}}, \mathbf{z_{2.4}}}$)

fils($x_{1.1}, y_{1.1}, z_{1.1}$; $x_{1.2}, y_{1.2}, z_{1.2}$; $x_{1.3}, y_{1.3}, z_{1.3}$; $x_{2.4}, y_{2.4}, z_{2.4}$)

La deuxième stratégie est le croisement uniforme UX (de l'anglais *Uniform Crossover*). Il effectue un croisement entre les atomes de deux parents avec un ratio de 50%. Les positions de croisement sont sélectionnées aléatoirement.

Exemple du UX :

Croisement(parent1 ;parent2 ;sélection{0 ;1 ;1 ;0})

Parent1($\underline{\mathbf{x_{1.1}}, \mathbf{y_{1.1}}, \mathbf{z_{1.1}}}$; $x_{1.2}, y_{1.2}, z_{1.2}$; $x_{1.3}, y_{1.3}, z_{1.3}$; $\underline{\mathbf{x_{1.4}}, \mathbf{y_{1.4}}, \mathbf{z_{1.4}}}$)

Parent2($x_{2.1}, y_{2.1}, z_{2.1}$; $\underline{\mathbf{x_{2.2}}, \mathbf{y_{2.2}}, \mathbf{z_{2.2}}}$; $\underline{\mathbf{x_{2.3}}, \mathbf{y_{2.3}}, \mathbf{z_{2.3}}}$; $x_{2.4}, y_{2.4}, z_{2.4}$)

fils($x_{1.1}, y_{1.1}, z_{1.1}$; $x_{2.2}, y_{2.2}, z_{2.2}$; $x_{2.3}, y_{2.3}, z_{2.3}$; $x_{1.4}, y_{1.4}, z_{1.4}$)

4.2.1.3 Opérateur de mutation

La mutation utilisée est une RL qui permet à un atome aléatoire de se déplacer dans l'espace fractionnel. La position atomique est ici définie avec ses coordonnées dans l'espace fractionnelles, ce qui veut dire que les bords de l'UC sont utilisés comme vecteurs de base pour décrire la position atomique, l'UC étant un parallélépipède défini par les longueurs de ses bords a, b, c et les angles entre eux a, β, et γ comme le montre la figure 4.3. Des individus sont choisis pour faire la RL, ces individus sont selectionnés grâce a une probabilité. Un atome aléatoire pour chaque individu sélectionné est déplacé selon un *pas* dans l'espace fractionnel. L'atome est déplacé le long des axes (x, y, z) et l'algorithme maintient le meilleur mouvement. Cette RL représente **l'hybridation à bas niveau** du P-GHAZ. La figure 4.8 montre la RL d'un individu après avoir effectué son croisement et l'algorithme 18 présente le pseudo-code de cette RL.

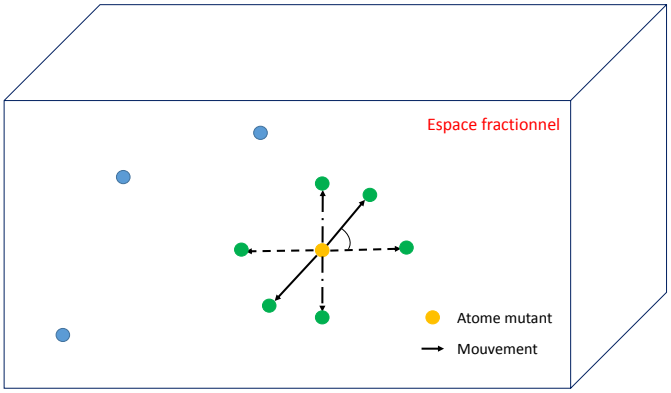


FIGURE 4.8 – Représentation de la RL d'un individu.

Algorithme 18 pseudo-code de la mutation

Entrées : *step* : le pas qui permet de déplacer l'atome mutant ; *ObjF* : la fonction objectif ; *mutantAtom* : l'atome mutant ; *mutantIndividual* : l'individu mutant
Sorties : *currentFitness* : la fitness courante de l'individu mutant ; *bestFitness* : la meilleure fitness générée par tous les mouvements
bestFitness := *currentFitness* ;
pour *coord* := 1 à 3 **faire** /* les coordonnées **x, y, z** */
    Ajouter *step* à la position *coord* du *mutantAtom* ;
    Evaluer *currentFitness* du *mutantIndividual* avec *ObjF* ;
    si *currentFitness* < *bestFitness* **alors**
        bestFitness := *currentFitness* ;
        Enregistrer le mouvement de *coord* ;
    fin si
    Enlever 2 × *step* à la position *coord* du *mutantAtom* ;
    Evaluer *currentFitness* du *mutantIndividual* avec *ObjF* ;
    si *currentFitness* < *bestFitness* **alors**
        bestFitness := *currentFitness* ;
        Enregistrer le mouvement de *coord* ;
    fin si
fin pour

4.2.2 L'algorithme MEGA-HZ

La deuxième proposition pour résoudre le problème des structures zéolitiques est un AG hybridé avec une heuristique basée sur la mémorisation et la prédiction des positions d'atomes. Nous avons nommé cette proposition MEGA-HZ (de l'anglais *MEmory Genetic Algorithme Hybridized for Zeolite*). Pour créer une zéolithe potentielle par le MEGA-HZ, la même modélisation des individus et la même procédure d'évaluation sont utilisées.

Deux variantes parallèles sont proposées, la première utilise **le niveau de parallélisation des individus** et la seconde variante ajoute à ce niveau **le niveau de parallélisation des algorithmes**. Pour les deux niveaux, la bibliothèque EASEA gère le parallélisme sur le GPU et sur le cluster.

Le MEGA-HZ génère une population aléatoire. La représentation des individus et leurs sélections sont effectuées exactement comme dans l'algorithme P-GHAZ. Les atomes d'oxygène sont toujours ajoutés à la fin de l'algorithme au centre des deux atomes T susceptibles d'être reliés. La RL et l'évaluation sont celles du premier algorithme également. Seul le croisement et l'heuristique de prédiction de la mémoire affichent des nouveautés.

4.2.2.1 Opérateurs de croisement

Pour l'algorithme MEGA-HZ, nous avons mis en place trois stratégies de croisement. Les deux premières stratégies sont déjà utilisées avec le P-GHAZ et nommées OPX et UX. La troisième stratégie est appelée RX (de l'anglais *Random crossover*). Cette stratégie tire deux valeurs aléatoires, *r1* et *r2*, pour chaque position. Si $r1 > r2$ l'algorithme prend l'atome du parent1 sinon il prend celui du parent2. La suite est un exemple du RX avec $r1 > r2$ pour la première position et $r1 < r2$ pour les autres positions :

Croisement(parent1 ;parent2 ;RX)

Parent1($\underline{\mathbf{x_{1.1}}, \mathbf{y_{1.1}}, \mathbf{z_{1.1}}}; x_{1.2}, y_{1.2}, z_{1.2}; x_{1.3}, y_{1.3}, z_{1.3}; x_{1.4}, y_{1.4}, z_{1.4}$)

Parent2($x_{2.1}, y_{2.1}, z_{2.1}; \underline{\mathbf{x_{2.2}}, \mathbf{y_{2.2}}, \mathbf{z_{2.2}}}; \underline{\mathbf{x_{2.3}}, \mathbf{y_{2.3}}, \mathbf{z_{2.3}}}; \underline{\mathbf{x_{2.4}}, \mathbf{y_{2.4}}, \mathbf{z_{2.4}}}$)

fils($x_{1.1}, y_{1.1}, z_{1.1}; x_{2.2}, y_{2.2}, z_{2.2}; x_{2.3}, y_{2.3}, z_{2.3}; x_{2.4}, y_{2.4}, z_{2.4}$)

4.2.2.2 Prédiction de la mémoire

Cette heuristique se base sur la mémorisation et la prédiction. Chaque fois que l'évaluation de la structure est améliorée, l'algorithme mémorise le mouvement que chaque atome à réaliser dans la structure. Cette mémorisation est effectuée en calculant la différence des coordonnées entre les anciennes positions et celles qui sont nouvelles. Au bout d'un certain nombre de mémorisations (trois dans notre proposition), l'algorithme prédit la prochaine position d'amélioration des atomes. Cette prédiction est réalisée en faisons une moyenne pondérée des derniers mouvements mémorisés. Le dernier mouvement mémorisé est celui qui compte le plus dans la prédiction. Le signe de la valeur de prédiction définit la direction de la prédiction. Après cette action, la mémoire est effacée et la procédure est répétée. Comme pour la mutation, les atomes évoluent dans l'espace fractionnel. La figure 4.9 illustre la prédiction d'un atome. L'algorithme 19 présente le pseudo-code de la première variante de l'algorithme MEGA-HZ et l'algorithme 20 présente le pseudo-code de l'heuristique de prédiction de la mémoire.

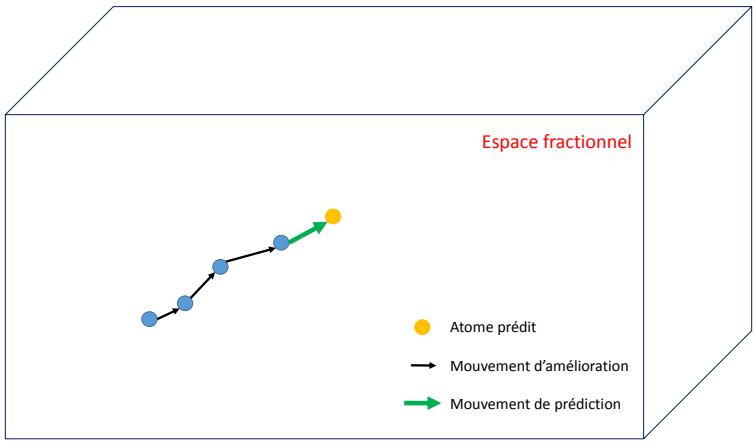


FIGURE 4.9 – Illustration d'une prédiction d'un atome après trois mémorisations.

Algorithme 19 Le pseudo-code de MEGA-HZ

Entrées : *initAtoms* : le nombre total des atomes initiaux dans l'unité asymétrique ; N : la taille de la population ; *step* : la valeur du pas qui est effectué par l'atome dans la mutation ; *ObjF* : la fonction objectif
Sorties : *currentFitness* : la fitness de l'individu courant ; *atome* : un individu de la population
/* Initialisation */
pour tous les N individus de la population en parallèle **faire**
    pour $i := 1$ à *initAtoms* **faire**
        Générer une position aléatoire **x** de *atom(i)* à l'intérieur des bornes de l'UC ;
        Générer une position aléatoire **y** de *atom(i)* à l'intérieur des bornes de l'UC ;
        Générer une position aléatoire **z** de *atom(i)* à l'intérieur des bornes de l'UC ;
    fin pour
fin pour
répéter
    /* Sélection */
    pour tous les N individus de la population en parallèle **faire**
        Sélectionner aléatoirement un parent de la population ;
    fin pour
    /* Croisement */
    pour tous les N individus de la population en parallèle **faire**
        Croisement entre l'individu courant et le parent sélectionné ;
    fin pour
    /* Prédiction de la mémoire */
    pour tous les N individus de la population en parallèle **faire**
        Appliquer l'heuristique de prédiction de la mémoire (voir algorithme 20) ;
    fin pour
    /* Mutation */
    Sélectionner aléatoirement un pourcentage prédéfini d'individus mutants ;
    pour tous les individus mutants en parallèle **faire**
        Sélectionner aléatoirement un atome mutant de l'individu mutant ;
        RL de l'atome mutant selon la valeur *step* (voir algorithme 18) ;
    fin pour
    /* Evaluation */
    pour tous les N individus de la population en parallèle **faire**
        Evaluer la *currentFitness* de l'individu avec *ObjF* ;
    fin pour
jusqu'à ce que les conditions d'arrêt soient satisfaites

Algorithme 20 pseudo-code de l'heuristique de prédiction de la mémoire

Entrées : *currentFitness* : la fitness courante ; *Memocompteur* : le compteur de mémorisation qui définit l'instant de prédiction
Sorties : *bestFitness* : la meilleure fitness de l'individu ; nouvelle position des atomes
si *currentFitness* < *bestFitness* **alors**
    bestFitness := *currentFitness* ;
    Memocompteur++ ;
    Memoriser le mouvement des atomes ;
    si *Memocompteur* == 3 **alors**
        Memocompteur := 0 ;
        Prédiction de la prochaine position des atomes ;
        Réinitialisation de la mémorisation ;
    fin si
fin si

4.3 Experimentation de la formulation 1

4.3.1 Conditions expérimentales

Un ensemble de paramètres est utilisé dans notre protocole de test pour la formulation 1. Les algorithmes utilisent 32768 individus qui sont parallélisés avec GPU. EASEA gère la parallélisation du GPU où chaque individu est exécuté dans un thread. Trois conditions d'arrêt sont définies. La première fixe le nombre maximum de générations à 200. La seconde est satisfaite si la valeur de la fonction objectif optimale de zéro est atteinte. Dans ce cas, la structure est une topologie d'une zéolithe potentielle. La dernière condition est satisfaite si la meilleure solution trouvée n'est pas améliorée pendant 100 générations successives (critère de stagnation). La valeur du *pas* de la RL est réglée à 0,1 pour déplacer l'atome sélectionné. Seulement 2% de la population sont sélectionnés dans

chaque génération pour exécuter la RL. Le tableau 4.1 présente les paramètres utilisés dans la formulation 1.

TABLE 4.1 – Paramètres des algorithmes pour la formulation 1

Paramètre	**Valeur**
Taille de la population	32768
Nombre de génèrations	200
Pas de la RL	0,1
Probabilité de mutation	0,02
L1	60
L2	180
L	2,44
distance minimale entre atomes	2,8
distance maximale entre atomes	3,3

4.3.2 Jeu de tests des zéolithes cibles

L'idée principale de cette expérimentation consiste à utiliser un ensemble de paramètres particulier de zéolithes connues. Le premier objectif est de confirmer que nos algorithmes sont capables de retrouver la structure de la zéolithe cible à partir d'une initialisation aléatoire des positions des atomes T dans l'AU. Si une zéolithe cible est retrouvée, nous pouvons confirmer que notre fonction objectif modélise bien le problème. Cependant, avec cette approche, il y'a beaucoup de structures potentielles qui peuvent être générées avec les paramètres d'entrée et qui sont des optimums globaux selon la fonction objectif proposée, i.e. pour lesquelles la valeur de la fonction objectif est nulle. Le tableau 4.2 énumère trois topologies zéolitiques cibles avec leur paramètres correspondants. *NbGrpEs* est le numéro du groupe d'espace, *NbOpSym* est le nombre d'opérations de symétrie correspondant au groupe d'espace, *NbAtomsAu* est le nombre initial des atomes T dans l'AU, et *NbAtomsUc* est le nombre total des atomes T évalués dans l'UC (après l'application des opérations de symétrie).

TABLE 4.2 – Zéolithes cibles

Topologies	**NbGrpEs**	**NbOpSym**	**NbAtomsAu**	**NbAtomsUc**
ITE	63	16	4	64
RTH	12	8	4	32
ITW	12	8	3	24

4.3.3 Validation numérique des topologies zéolitiques

La validation numérique a pour but de valider la capacité de nos algorithmes à satisfaire toutes les contraintes. Deux opérateurs de croisement sont utilisés dans l'algorithme P-GHAZ, à savoir, l'OPX et l'UX. Un troisième opérateur, appelé RX, est expérimenté avec l'algorithme MEGA-HZ en plus des deux premiers. La complexité de l'évaluation dépend du paramètre *NbAtomsUc*. Plus ce nombre est élevé, plus l'algorithme passe du temps dans l'évaluation.

L'algorithme exécute le croisement entre les positions générées dans l'AU. Le tableau 4.3 présente la fitness (F) et le temps d'exécution de nos algorithmes sur les paramètres des trois topologies. Un ensemble de 10 tentatives est exécuté pour chaque instance et le temps d'exécution est exprimé en minutes. La moyenne du temps de calcul et de la fitness est présentée ainsi que *NB* qui représente le nombre de fois où l'algorithme atteint l'optimalité.

4.3.3.1 Résultats de l'algorithme P-GHAZ

Le tableau 4.3 présente les résultats des deux opérateurs de croisement appliqués par le P-GHAZ. Nous pouvons observer que l'opérateur OPX a la capacité d'obtenir la valeur optimale pour les trois zéolithes cibles du jeu de tests. UX ne peut pas obtenir des résultats optimaux sur la topologie **RTH**. Cependant, UX est plus efficace avec la topologie **ITE**. D'une manière générale, nous pouvons observer que le P-GHAZ peut obtenir la valeur optimale de zéro dans de nombreux essais avec ces deux opérateurs de croisement. Avec

la formulation 1 de la fonction objectif, le P-GHAZ atteint l'optimalité dans 33 tentatives parmi les 60 réalisées, soit un taux de succès de 55%. Le meilleur taux, égal à 60%, est obtenu par UX.

TABLE 4.3 – Résultats obtenus sur le jeu de tests des zéolithes

tentatives	**ITE**				**RTH**				**ITW**			
	OPX		**UX**		**OPX**		**UX**		**OPX**		**UX**	
	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps
1	16	60,2	**0**	32,6	1	19,9	4	14,7	**0**	0,9	**0**	1,5
2	9	71,9	19	65,1	**0**	5,3	4	18,3	**0**	1	**0**	0,8
3	8	80,4	**0**	31,2	2	19,3	4	15,9	**0**	0,8	**0**	1,7
4	8	88,5	**0**	26,8	**0**	4,3	4	17,9	**0**	1,2	**0**	1,6
5	8	60,4	**0**	29,5	1	21,1	4	17,9	**0**	0,7	**0**	1,1
6	9	55,7	**0**	28,2	1	18,7	4	15	**0**	1	**0**	1,2
7	2	83,4	17	77,8	**0**	3,6	4	15,3	**0**	1,3	**0**	1,1
8	8	55,8	**0**	20,1	**0**	9,2	5	14,5	**0**	1	**0**	1,3
9	**0**	24,4	**0**	27,8	3	21,3	5	15,7	**0**	0,9	**0**	1,3
10	16	54,8	**0**	32,1	1	19,5	6	14,9	**0**	1,4	**0**	0,9
Moyenne	8,4	63,5	3,6	37,1	0,9	14,2	4,4	16	0	1	0	1,3
NB	1		8		4		0		10		10	

4.3.3.2 Résultats de l'algorithme MEGA-HZ

Le tableau 4.4 montre les résultats des trois opérateurs de croisement utilisés par le MEGA-HZ. L'opérateur OPX conserve sa capacité à obtenir la valeur optimale pour les trois jeu de tests de zéolithes. L'opérateur RX obtient l'optimalité beaucoup plus souvent que les autres croisements (OPX et UX). UX ne peut pas obtenir de résultats optimaux sur la topologie **RTH**, ni sur la topologie **ITE**. Si on considère seulement les opérateurs UX et OPX, le MEGA-HZ atteint l'optimalité dans 19 tentatives parmi 60, soit un taux de 32%. L'OPX reste toujours efficace avec un taux de 47%. Le RX est largement devant avec un taux de 73% soit 22 tentatives sur 30. Globalement, le MEGA-HZ obtient 41 tentatives optimales sur 90, soit un taux de 46%.

TABLE 4.4 – Résultats obtenus sur le jeu de tests des zéolithes

tentatives	ITE						RTH						ITW					
	OPX		UX		RX		OPX		UX		RX		OPX		UX		RX	
	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps
1	8	62,6	30	27,5	3	78,5	**0**	6	8	7,1	**0**	5,2	**0**	1,6	1	6,5	**0**	1,1
2	4	77,3	17	28,7	**0**	14,7	**0**	6,4	9	11,2	2	18,5	**0**	1	1	6,8	**0**	1,4
3	4	68	26	56,4	**0**	8	2	18,7	5	6,7	**0**	4,8	**0**	1,1	1	7,2	**0**	1,4
4	8	65	32	39,1	**0**	17,3	2	19	7	15,1	1	18,8	**0**	2,1	1	8,4	**0**	0,5
5	1	70,2	20	37,7	2	107,3	1	19	9	13,5	**0**	5,4	**0**	1,3	**0**	1,3	**0**	1,1
6	2	74,3	26	63,7	**0**	17,6	2	18,4	9	13,6	2	21	**0**	1,6	**0**	0,8	**0**	1,2
7	4	72,8	23	33,9	7	70,9	1	21,6	11	11	**0**	4,2	**0**	1,5	**0**	1,9	**0**	1,3
8	4	87,6	32	42,1	**0**	18,9	**0**	8,6	4	11,5	1	19,1	**0**	1,6	**0**	0,2	**0**	0,3
9	**0**	13,7	27	27,7	**0**	39,3	1	17,8	13	6,7	1	19,3	**0**	1,8	1	6,4	**0**	2,1
10	8	78,1	28	24,6	**0**	16,5	1	18,5	7	13,1	**0**	4,7	**0**	0,8	**0**	1,14	**0**	1,6
Moyenne	4,3	67	26,1	38,1	1,2	38,9	1	15,4	8,2	10,9	0,7	12,1	0	1,5	0,5	4,1	0	1,2
NB	1		0		7		3		0		5		10		5		10	

4.3.4 Validation des structures zéolitiques obtenues par le P-GHAZ

La validation des structures zéolitiques se fait en deux étapes. La première étape est la validation visuelle que nous effectuons pour les zéolithes que l'algorithme génère avec une valeur optimale. La deuxième étape s'effectue dans le laboratoire de chimie pour le calcul de l'énergie et de la stabilité de la structure.

Les structures avec une évaluation optimale sélectionnées doivent être validées à la deuxième étape dans le laboratoire de chimie. Pour cela, les meilleurs candidats sont minimisés, à l'aide de logiciels spécialisés, afin de placer correctement les atomes d'oxygène et de silicium dans une configuration acceptable. Ce placement se fait sans tenir compte des interactions électrostatiques de modélisation moléculaire en utilisant le champ de force universelle (UFF de l'anglais *Universal Force Field*) (Rappe et al., 1992) mis en oeuvre dans le logiciel *Cerius*^{2} (Cerius^2, 2000).

À travers les trois jeux de tests de zéolithes testés dans la formulation 1 avec le P-GHAZ, nous avons sélectionné 42 structures pour la deuxième étape de validation pour le laboratoire de chimie. Le P-GHAZ a réussi à obtenir six topologies possibles qui peuvent être redondantes dans les 42 structures envoyées. Trois d'entre elles sont déjà connues et les trois autres sont nouvelles. Pour les trois dernières, une meilleure optimisation est ensuite effectuée en utilisant le *non-polarizable Bushuev-Sastre (BS) force field* capable de reproduire avec une grande précision la structure et l'énergie des zéolithes de silice pure (Bushuev & Sastre, 2009).

Finalement, l'optimisation de la géométrie des structures des zéolithes hypothétiques examinées dans une forme purement siliceuse est réalisée à l'aide du programme *General Utility Lattice Program (GULP)* (Gale, 1997).

4.3.4.1 Les résultats à partir des paramètres de ITE

À partir des paramètres de la topologie **ITE**, le P-GHAZ a trouvé trois zéolithes viables. Comme représentée sur la figure 4.10, la topologie **ITE** elle-même a été trouvée.

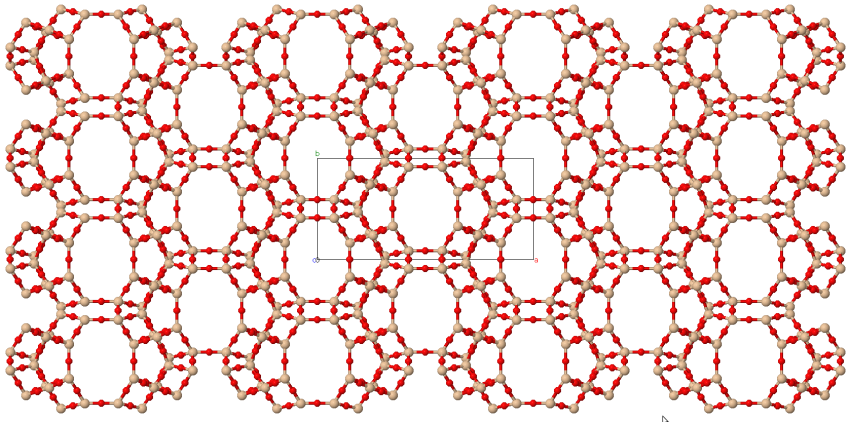


FIGURE 4.10 – La topologie **ITE** retrouvée par P-GHAZ (avant la minimisation (UFF) et avant l'optimisation (BS)).

La deuxième topologie trouvée est différente de **ITE**. Elle correspond à la topologie **MER** qui est normalement décrite dans le groupe d'espace tétragonal *I4/mmm* (le **NbGrpEs** est 139, le **NbOpSym** est 32 et le **NbAtomsAu** est 1). Ce résultat prouve que notre fonction objectif est capable de trouver plusieurs structures différentes à partir d'un ensemble de paramètres tels que les paramètres **ITE**. La figure 4.11 montre la zéolithe **MER** trouvée par notre algorithme.

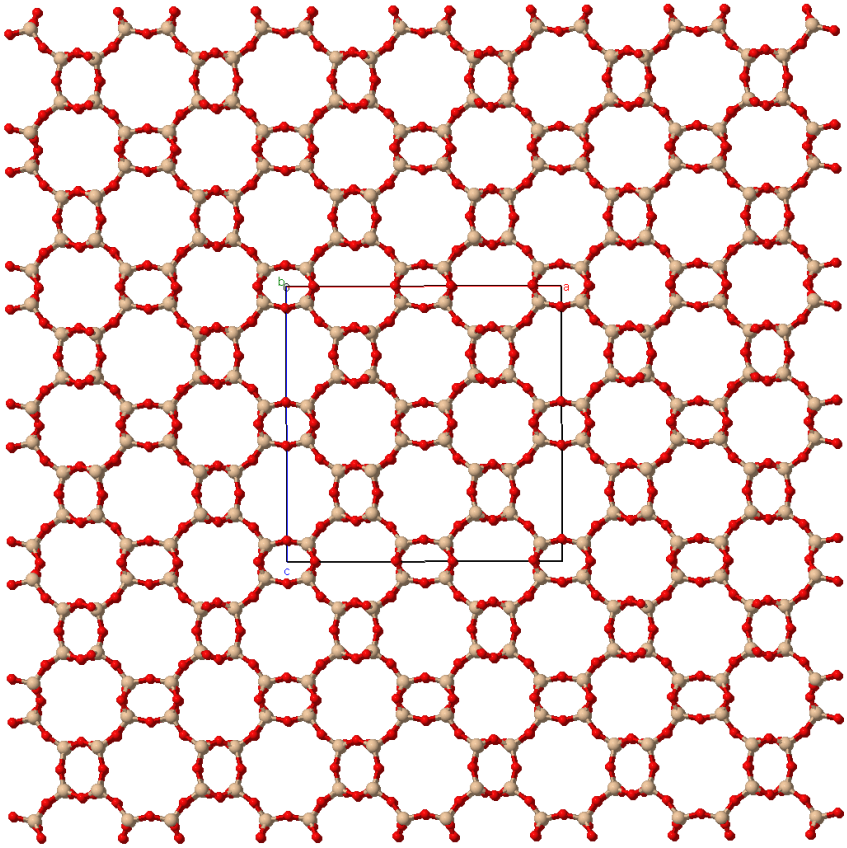


FIGURE 4.11 – La topologie **MER** retrouvée par P-GHAZ (avant la minimisation (UFF) et avant l'optimisation (BS)).

La troisième zéolithe stable trouvée par P-GHAZ est une nouvelle topologie qui n'est pas répertoriée sur le site Web du SC-IZA (Baerlocher & McCusker) ni dans l'*Atlas of*

Prospective Zeolite Structures (Foster & Treacy). Ce fait confirme le potentiel de notre formulation pour trouver des topologies inconnues. La figure 4.12 montre la nouvelle structure de zéolithe trouvée par P-GHAZ à partir des paramètres de **ITE**. Comme cela est illustré sur la figure 4.13, cette nouvelle topologie possède un système de pores à trois dimensions constitué d'un grand 12MR et de petits pores de 8MR. La structure de cette nouvelle topologie peut être décrite comme une combinaison d'unités de construction composites nommées CBU (de l'anglais *Composite Building Units*). Ainsi, comme illustré sur la figure 4.13, des double-4MR et des double-8MR (*d4r* et *d8r*) sont empilés le long de l'axe c pour former des colonnes (les axes a, b et c sont définis sur la figure 4.4). La structure complète est obtenue en reliant ces colonnes par des ponts oxygène formant des 4MR. Nous avons nommé cette nouvelle structure hypothétique $HZM\#1$ pour $HypotheticalZeoliteofMulhouse\#1$. Les nouvelles topologies zéolitiques suivantes seront nommées progressivement.

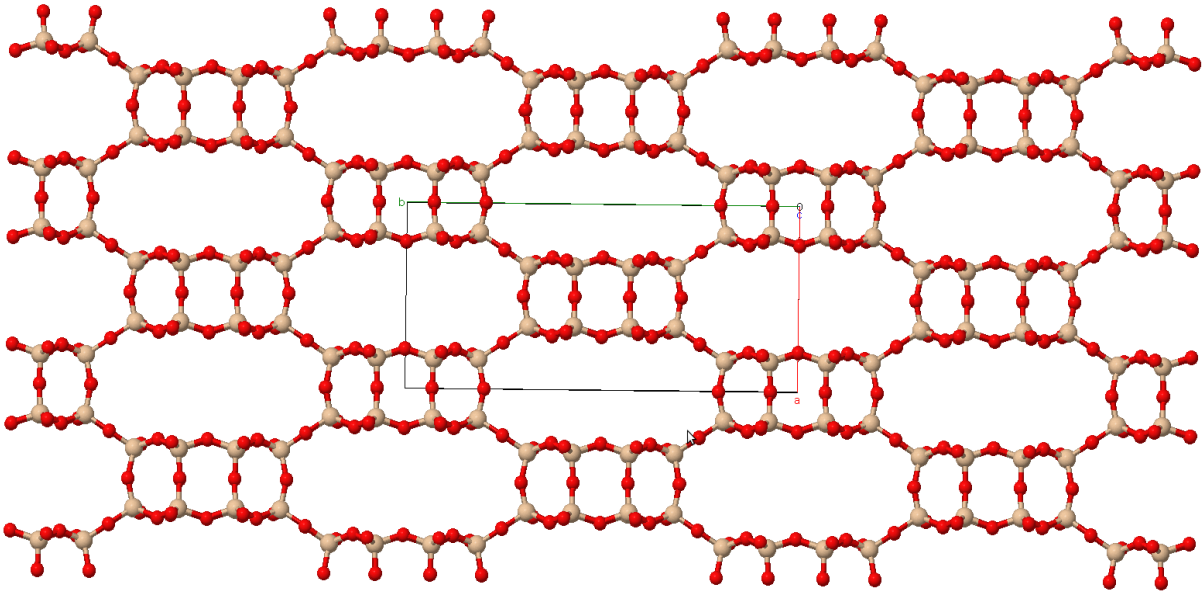
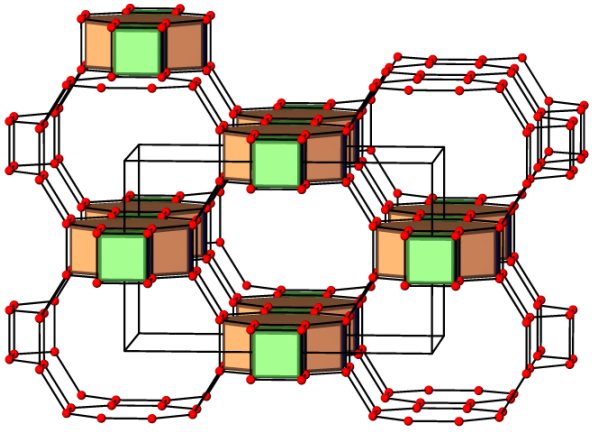
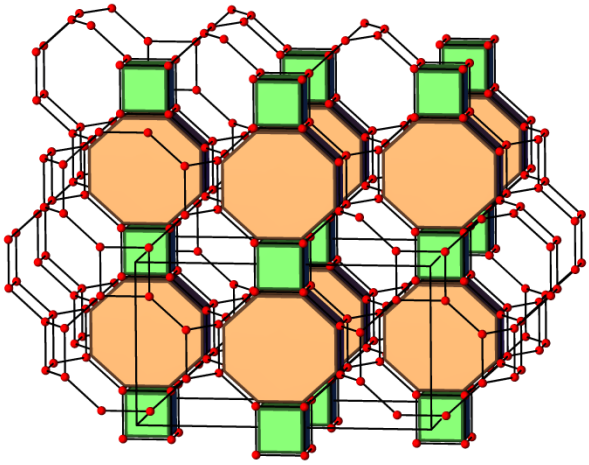


FIGURE 4.12 – La première nouvelle zéolithe $HZM\#1$ calculée par P-GHAZ (après la minimisation (UFF) et avant l'optimisation (BS)).



(a)



(b)

FIGURE 4.13 – La nouvelle topologie $HZM\#1$ trouvée par P-GHAZ après minimisation (UFF) et après optimisation (BS) (calculée avec ToposPro (Blatov et al., 2014) et intégrée avec 3dt (Delgado-Friedrichs, 2003)). Le *d4r* et le *d8r* sont en vert et brun, respectivement. Les atomes d'oxygène ont été omis pour des raisons de clarté.

4.3.4.2 Les résultats à partir des paramètres de ITW

À partir des paramètres de la topologie **ITW**, le P-GHAZ a trouvé deux topologies potentiellement viables. La zéolithe **ITW** n'a pas été trouvée, mais l'algorithme propose

d'autres structures proches de sa topologie. Malheureusement, ces structures ne sont pas viables, mêmes après la minimisation. Cependant, de manière inattendue, la topologie **CHA**, une zéolithe classée dans le SC-IZA, a été trouvée. Il est intéressant de noter que le vrai groupe d'espace pour cette topologie est $R\bar{3}m$ (le **NbGrpEs** 166, le **NbOpSym** est 36 et le **NbAtomsAu** est 1). La figure 4.14 montre la zéolithe **CHA** trouvée par P-GHAZ.

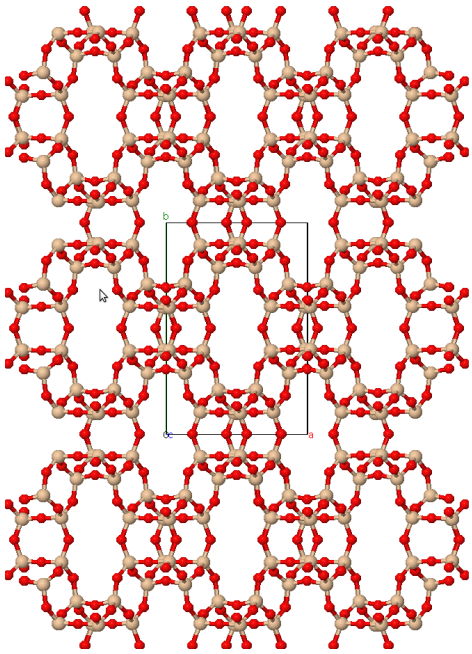


FIGURE 4.14 – La topologie **CHA** retrouvée par P-GHAZ dans le groupe d'espace *C2/m* (après la minimisation (UFF) et avant l'optimisation (BS)).

La seconde nouvelle zéolithe ($HZM\#2$) trouvée par P-GHAZ, à partir des paramètres de **ITW**, n'est pas répertoriée dans les bases de données. Elle est représentée sur la figure 4.15. Comme cela est illustré sur la figure 4.16, cette nouvelle topologie possède un système de pores à trois dimensions constitué de grands pores 12MR et de petits pores de 8MR.

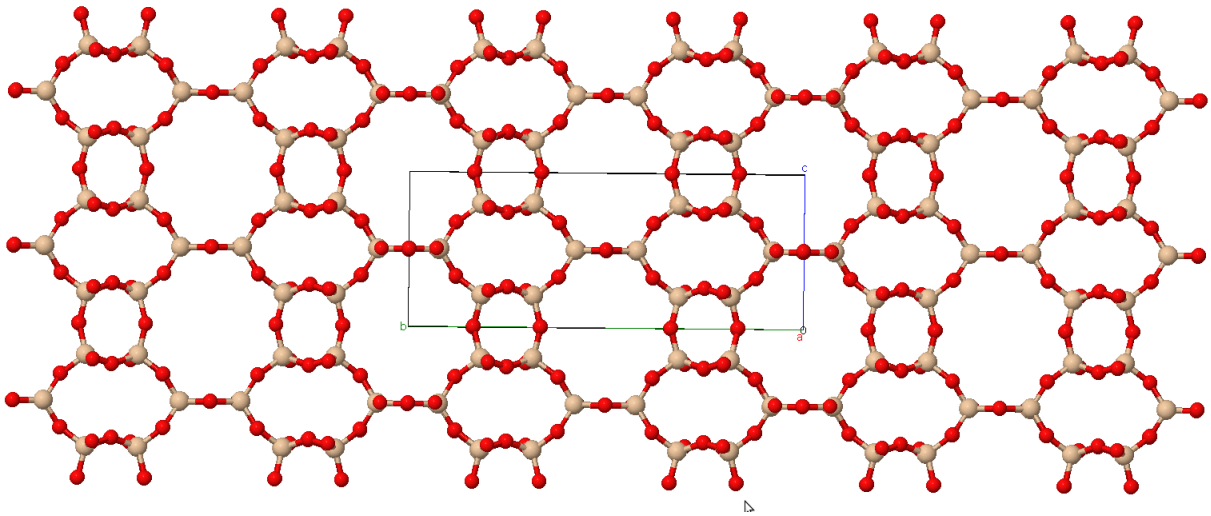


FIGURE 4.15 – La deuxième nouvelle zéolithe ($HZM\#2$) calculée par P-GHAZ après la minimisation (UFF) et avant l'optimisation (BS).

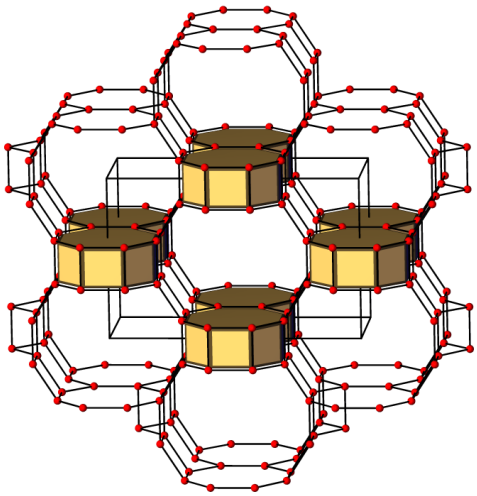


FIGURE 4.16 – La deuxième nouvelle topologie $HZM\#2$ trouvée par P-GHAZ après minimisation (UFF) et après optimisation (BS). Les CBUs *d4r* sont absents entre les unités *d8r* par rapport à la première topologie inconnue décrite ci-dessus. Les atomes d'oxygène ont été omis pour des raisons de clarté.

4.3.4.3 Les résultats à partir des paramètres de RTH

La topologie **RTH** n'a pas été trouvée à partir de ses paramètres. D'autres topologies proches ont été générées mais qui ne sont pas stables énergétiquement. Cependant, une nouvelle structure de zéolithe potentielle ($HZM\#3$) a émergé des calculs. Sur la figure 4.17 nous pouvons voir la troisième nouvelle zéolithe trouvée par P-GHAZ à partir des paramètres de **RTH** avant la minimisation de l'énergie. Comme illustré sur la figure 4.18, $HZM\#3$ possède un système de pores à deux dimensions constitué de petits pores 8MR. Dans cette topologie, la connectivité entre les tétraèdres de silicium conduit à la formation d'un nouveau CBU (figure 4.18a). Il est formé par quatre pores 4MR et quatre pores 6MR, conçus sous la forme $[4^4 6^4]$.

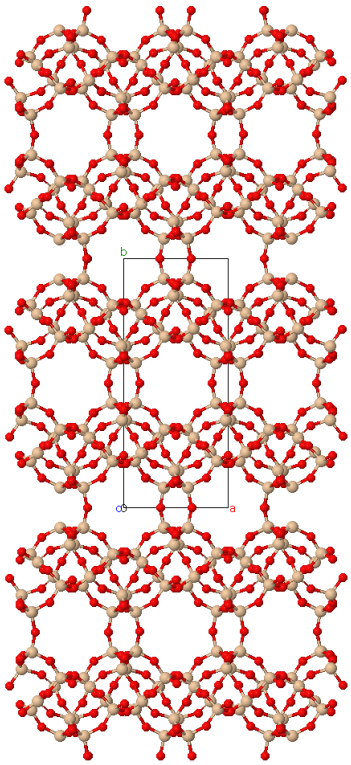
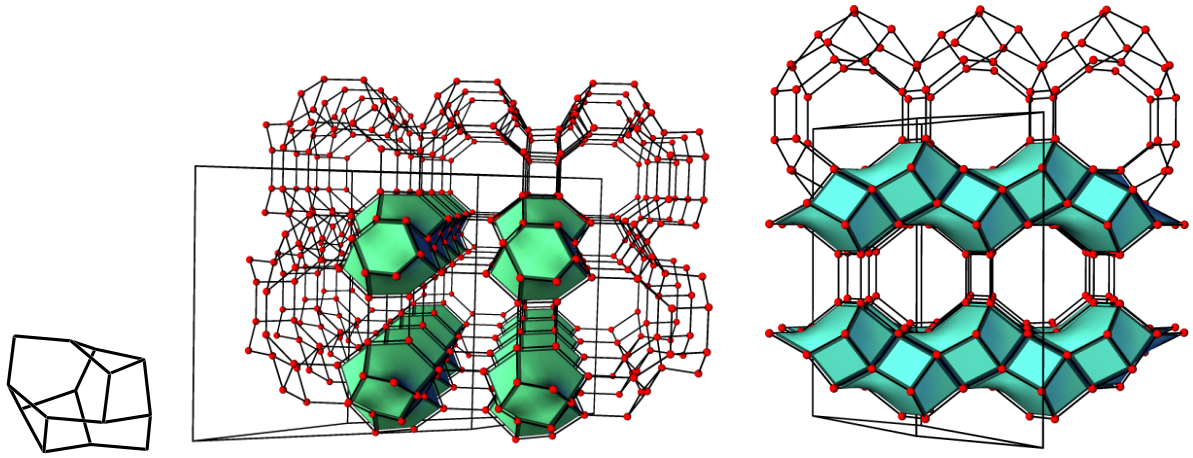


FIGURE 4.17 – La troisième nouvelle zéolithe calculée ($HZM\#3$) avec P-GHAZ à partir des paramètres de **RTH** après la minimisation (UFF) et avant l'optimisation (BS).



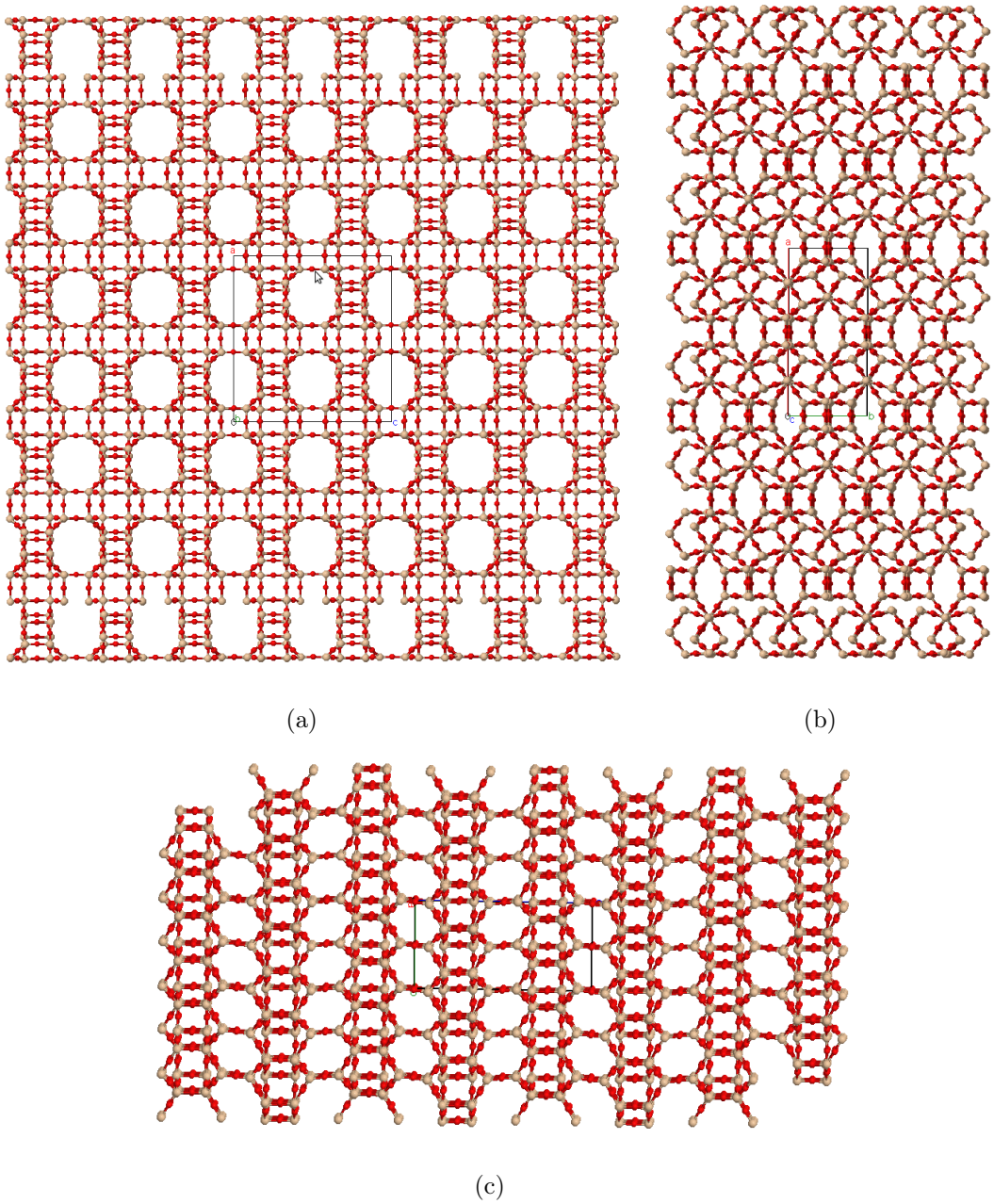
(a) (b) (c)

FIGURE 4.18 – (a) le nouveau CBU $[4^4 6^4]$ observé dans $HZM\#3$ et trouvé par P-GHAZ, après la minimisation (UFF) et après optimisation (BS). Les colonnes des CBUs $[4^4 6^4]$ sont en bleu azur. Les atomes d'oxygène ont été omis pour des raisons de clarté.

4.3.5 Sélection des structures zéolitiques obtenues par MEGA-HZ

Pour l'algorithme MEGA-HZ, seule la première étape de validation est effectuée. La deuxième étape de validation est en cours dans le laboratoire de chimie. Du coté informatique, la sélection se fait surtout en regardant les pores formés par la structure et la formation de CBU dans ces sélections. Pour la formulation 1, avec l'algorithme MEGA-HZ, nous avons sélectionné 15 structures à partir des 90 tentatives réalisées pour la validation numérique. Les figures suivantes proposent une topologie sélectionnée par zéolithe cible.

À partir des paramètres de la topologie **ITE**, la figure 4.19 montre une structure trouvée par MEGA-HZ avec l'opérateur de croisement OPX.

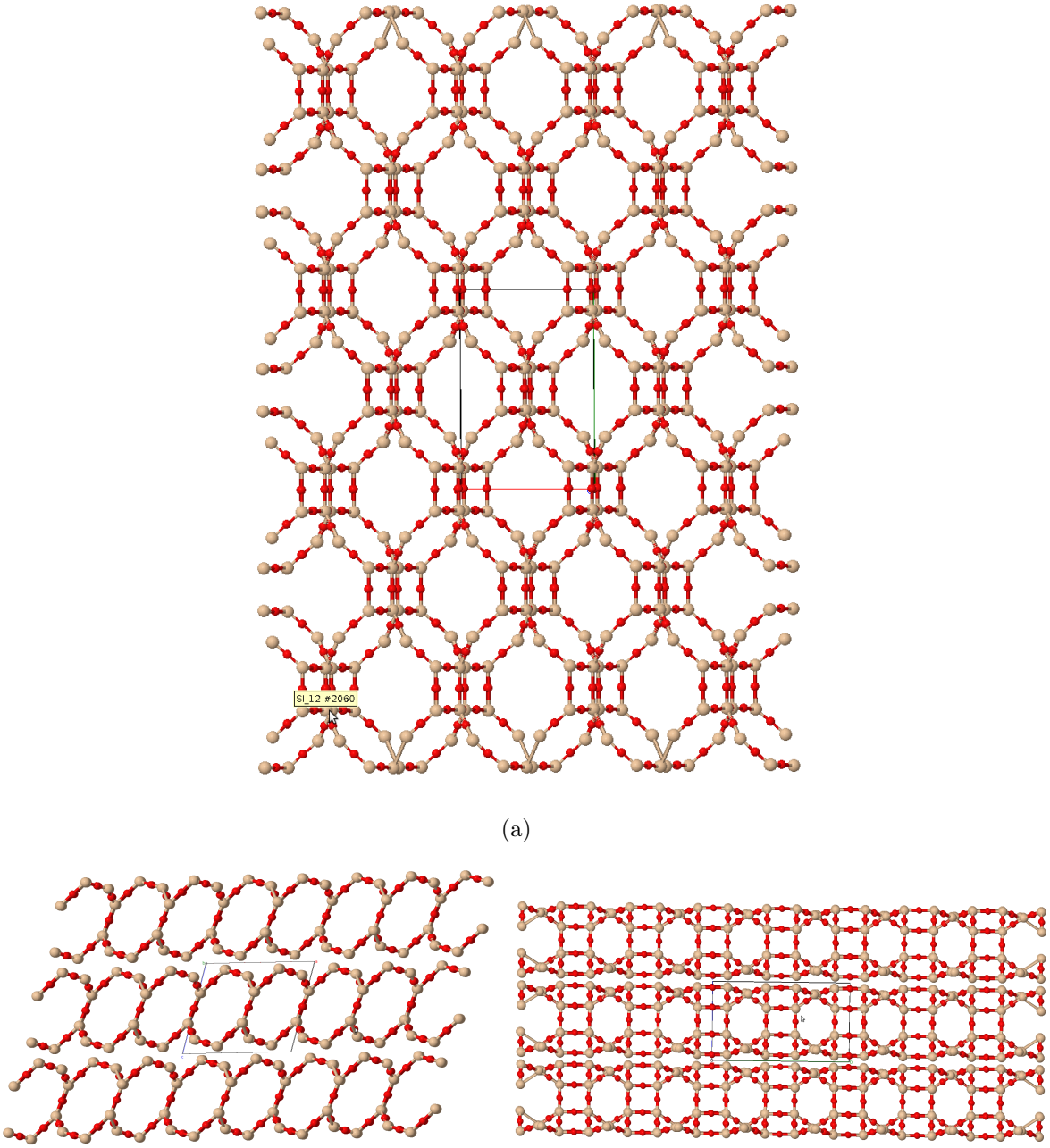


(a) (b)

(c)

FIGURE 4.19 – Première zéolithe sélectionnée avec MEGA-HZ à partir des paramètres de **ITE** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **ITW**, la figure 4.20 montre une structure trouvée par le MEGA-HZ avec l'opérateur de croisement RX.



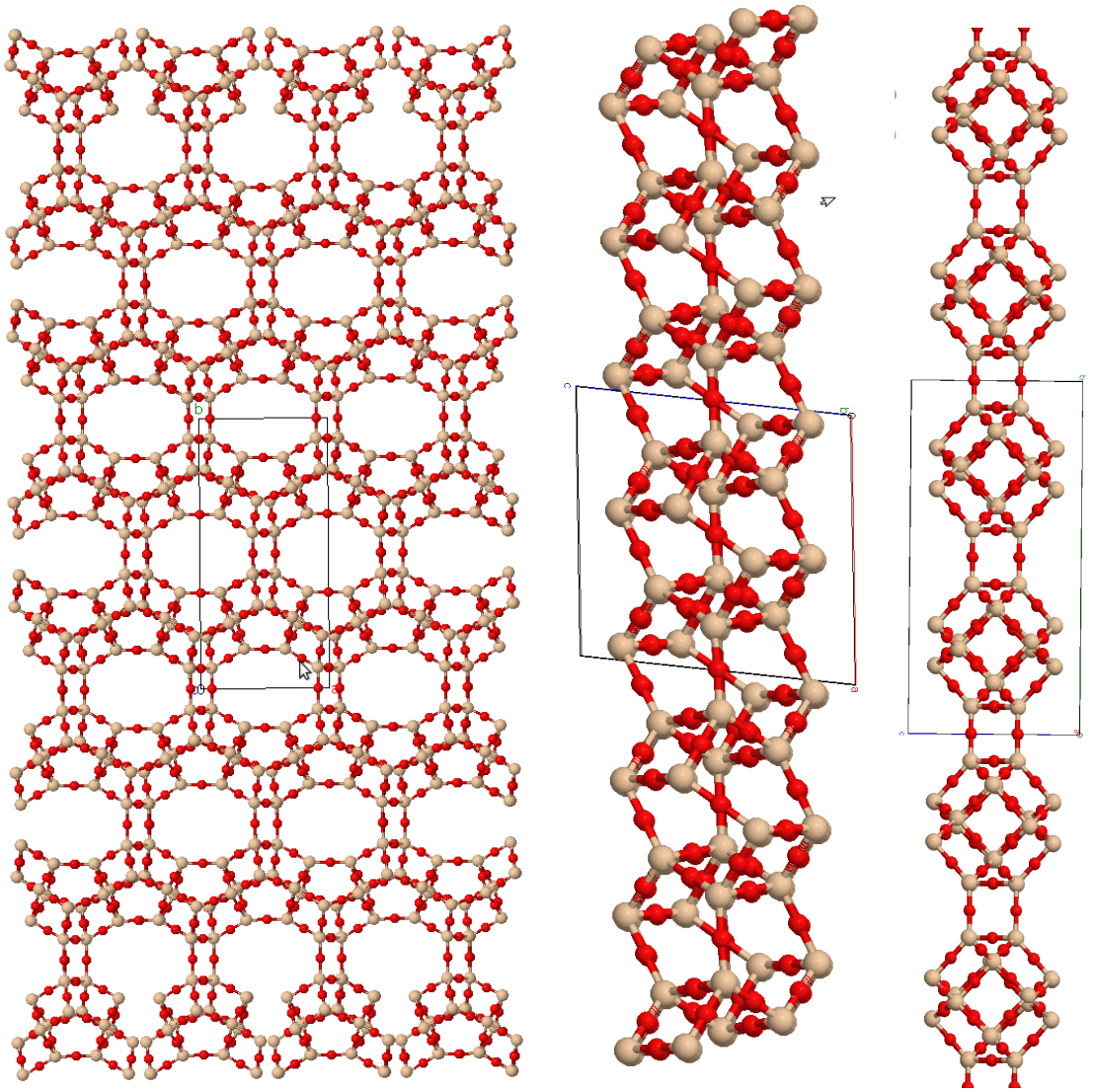
(a)

(b)

(c)

FIGURE 4.20 – Deuxième zéolithe sélectionnée avec MEGA-HZ à partir des paramètres de **ITW** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **RTH**, la figure 4.21 montre une structure trouvée par le MEGA-HZ avec l'opérateur de croisement UX.



(a) (b) (c)

FIGURE 4.21 – Troisième zéolithe sélectionnée avec MEGA-HZ à partir des paramètres de **RTH** avant la minimisation (UFF) et avant l'optimisation (BS).

4.4 Experimentation de la formulation 2

4.4.1 Conditions expérimentales

Un ensemble de paramètres est utilisé dans notre protocole de test pour la formulation 2 de la fonction objectif. Le MEGA-HZ est le seul algorithme testé avec cette formulation. Le niveau de parallélisation des algorithmes est ajouté dans cette expérimentation. Plusieurs îlots sont expérimentés en parallèle avec un échange d'informations périodique entre les îlots. Le MEGA-HZ utilise 32768 individus par îlots qui sont parallélisés avec GPU. La plateforme EASEA gère la parallélisation sur GPU, où chaque individu est exécuté dans un thread. EASEA gère également la distribution des îlots sur 8 processus, où chaque processus occupe une machine de notre cluster. Après chaque génération, chaque processus envoie sa meilleure solution vers son processus voisin et cette solution est intégrée à la population voisine.

Trois conditions d'arrêt sont définies. La première fixe le nombre maximum de générations à 1000. La seconde est déclenchée si la valeur de la fonction objectif est inférieure ou égale à zéro. Dans ce cas, la structure est une topologie d'une zéolithe potentielle. La dernière condition est déclenchée si la meilleure solution trouvée n'est pas améliorée pendant 250 générations successives (critère de stagnation dur). Lorsque la meilleure solution trouvée n'est pas améliorée pendant 100 générations successives (critère de stagnation souple), 60% de la population sont régénérés aléatoirement.

La valeur du *pas* de la RL pour déplacer l'atome sélectionné est aléatoire et différente pour les trois axes (x, y, z). Seulement 2% de la population sont sélectionnés dans chaque génération pour exécuter la RL. Le tableau 4.5 présente les paramètres utilisés pour la formulation 2.

TABLE 4.5 – Paramètres de MEGA-HZ pour la formulation 2

Paramètre	**Valeur**
Taille de la population par îlot	32768
Nombre d'îlots	8
Nombre de génèration	1000
Probabilité de mutation	0,02
L1	60
L2	180
L	2,44
Distance atomes minimale	2,8
Distance atomes maximale	3,3

4.4.2 Jeu de tests des zéolithes cibles

Pour l'expérimentation de la formulation 2, nous gardons les trois topologies déjà présentées qui sont **ITE**, **ITW** et **RTH**. Nous ajoutons trois autres topologies qui sont **CFI**, **ISV** et **STF**. Le tableau 4.6 énumère ces trois topologies zéolitiques cibles avec leurs paramètres correspondants.

TABLE 4.6 – Zéolithes cibles

Topologies	**NbGrpEs**	**NbOpSym**	**NbAtomsAu**	**NbAtomsUc**
CFI	74	16	5	32
ISV	131	16	5	64
STF	12	8	5	32

4.4.3 Validation numérique des topologies zéolitiques

Les tableaux 4.7 et 4.8 montrent les résultats des trois opérateurs de croisement appliqués avec le MEGA-HZ distribué. Ces opérateurs sont OPX, UX et RX. Le tableau 4.7 affiche les résultats de **ITE**, **ITW** et **RTH** alors que le tableau 4.8 affiche ceux des topologies **CFI**, **ISV** et **STF**. Les valeurs qui sont inférieures ou égales à zéro sont considérées comme optimales. L'opérateur OPX continue à garantir au moins une tentative optimale pour toutes les six topologies. Avec cet algorithme, UX a le meilleur taux de réussite avec 44 tentatives optimales sur 60 tentatives, soit un taux de 73%. OPX obtient un taux de 62% de tentatives optimales sur les six topologies et finalement RX obtient un taux de 32% de tentatives optimales. Globalement, le MEGA-HZ distribué obtient 65 tentatives optimales sur les 90 du tableau 4.7, soit un taux de 72%, et 35 tentatives optimales sur les 90 du tableau 4.8, soit un taux de 39%.

TABLE 4.7 – Jeu de tests des zéolithes

tentatives	**ITE**						**RTH**						**ITW**					
	OPX		**UX**		**RX**		**OPX**		**UX**		**RX**		**OPX**		**UX**		**RX**	
	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps
1	16	172,4	**-9**	21	2	177,8	**-2**	6,1	**-1**	6,9	8	37,6	**0**	1,9	**-22**	1,3	**0**	1,9
2	**0**	17	**-7**	24,3	16	204,3	1	42,9	**-6**	6,6	8	37	**-1**	1,4	**-1**	1,2	**-11**	1,4
3	**-2**	16,2	**-37**	26,1	**-8**	55,6	**-6**	4,1	**0**	4,3	5	37,1	**-8**	1,3	**-6**	1,4	**-1**	1.4
4	**-3**	16,1	16	162,5	24	97,2	8	38,2	1	52,9	1	43,4	**-1**	1,2	**0**	0,9	**0**	1,2
5	**-15**	40,8	**-5**	18,3	24	123,6	**0**	7,4	**-1**	4,1	1	53,4	**-1**	2,2	**-1**	0,9	**0**	2,2
6	**-15**	24,5	16	165,7	28	162,8	1	40	**-4**	7,4	**0**	17,9	**-10**	0,8	**0**	0,7	**-1**	0,8
7	16	167,7	**0**	133,8	2	170,3	**0**	8,1	**-3**	7	8	38,7	**-4**	1,4	**-1**	2,6	**0**	1,4
8	16	134,5	**-10**	21,7	**-8**	38,7	**0**	10,7	8	40.9	9	19,7	**-1**	1,2	**-6**	0,7	**-8**	1,2
9	**-4**	48,6	**-20**	82,6	**-17**	63,2	**0**	2,3	**0**	11,5	**-1**	9,8	**-5**	0,9	**-5**	0,4	**-3**	0,9
10	**-6**	21,2	**-20**	25,4	16	190	8	36,9	**0**	6,1	**-2**	5,3	**-1**	1,8	**0**	1,8	**-1**	1,8
Moyenne	0,3	65,9	-7,6	68,1	7,9	128,4	1	19,7	-0,6	14,8	3,7	30	-3,2	1,4	-4,2	1,2	-2,5	1,4
NB	7		8		3		6		8		3		10		10		10	

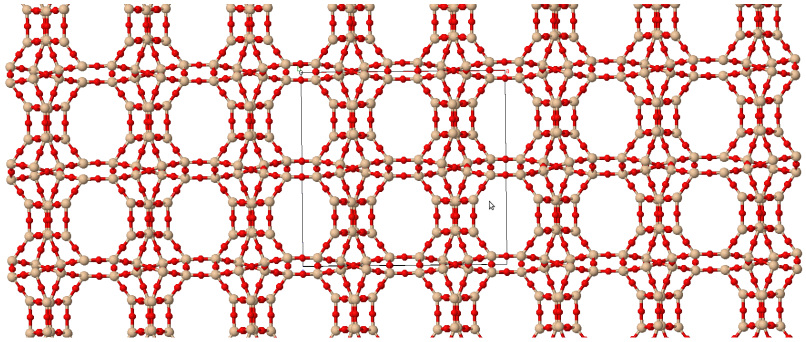
TABLE 4.8 – Jeu de tests des zéolithes

tentatives	CFI						STF						ISV					
	OPX		UX		RX		OPX		UX		RX		OPX		UX		RX	
	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps	F	temps
1	**-4**	10,1	**-16**	12,8	**-1**	12,9	5	47,4	**-2**	23,6	5	59,7	5	187	**-13**	45,3	41	38,1
2	**-1**	8,5	**-10**	14	8	50,5	2	75,8	4	41,5	8	53,4	5	195,9	**-8**	38,9	40	59,6
3	**0**	11	**-2**	16,2	8	35	**-12**	5,2	4	49,9	10	54	1	169,9	1	190,3	32	57,1
4	**0**	14,6	4	65,5	6	46	1	52,4	8	50,2	8	31,4	1	341,1	**-2**	171,8	21	71,2
5	3	67,6	**-4**	18,6	8	40,8	**-1**	11	4	58,7	**0**	14,5	**-16**	29,3	**-10**	96,8	45	34,7
6	4	59,6	**-4**	10,1	4	37,1	4	45,7	1	53,2	8	45,7	12	135,7	4	191,3	11	269,6
7	**-2**	10,7	**-5**	8,8	4	44,3	4	50	**-8**	18,3	16	17,5	5	170,2	**-7**	34	1	208,8
8	**-4**	15,9	**-1**	17,2	6	51,4	3	55	1	50,1	4	47,9	**-35**	28,8	**-10**	17,7	8	281,8
9	**0**	9,1	**-3**	9,2	**-1**	12,9	**-5**	7,6	**-3**	9,5	12	19,8	**-26**	79,5	**-2**	50,6	8	158,4
10	**0**	14,9	4	75,2	8	42,5	4	55,6	4	54,3	4	25,2	5	252,1	6	169,4	21	124,6
Moyenne	-0,4	22,2	-3,7	24,8	5	37,3	0,5	40,5	1,3	40,9	7,5	36,9	-4,3	158,9	-4,1	100,6	22,8	130,4
NB	8		8		2		3		3		1		3		7		0	

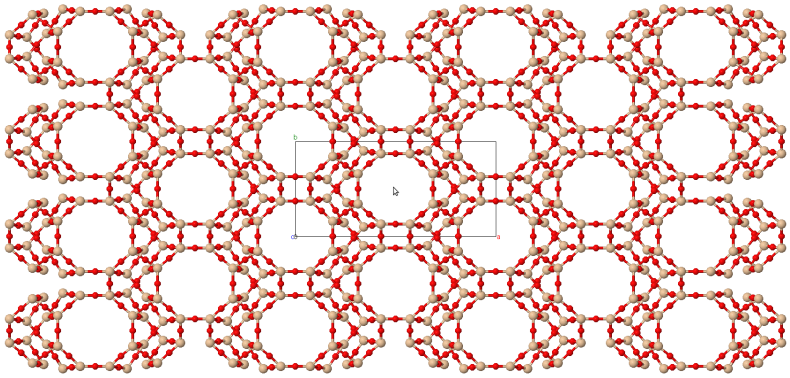
4.4.4 Sélection des structures zéolitiques

Pour l'algorithme MEGA-HZ distribué, seule la première étape de validation est effectuée. La deuxième étape est en cours de réalisation dans le laboratoire de chimie également. Pour la formulation 2, avec l'algorithme MEGA-HZ distribué, nous avons sélectionné 32 structures à partir des 180 tentatives réalisées pour la validation numérique. Nous vous proposons également de voir une topologie sélectionnée par zéolithe cible.

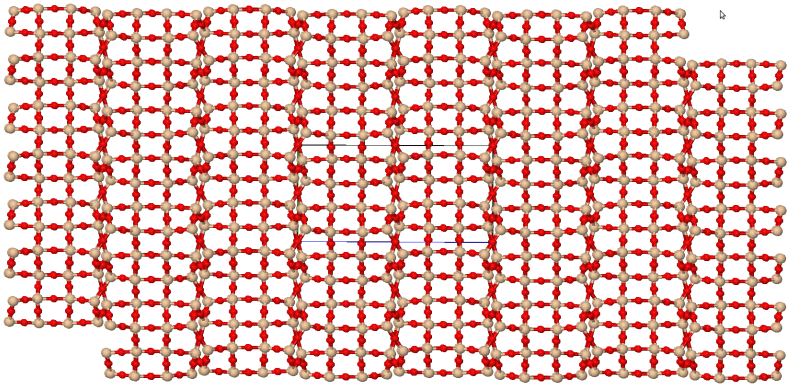
À partir des paramètres de la topologie **ITE**, la figure 4.22 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement OPX.



(a)



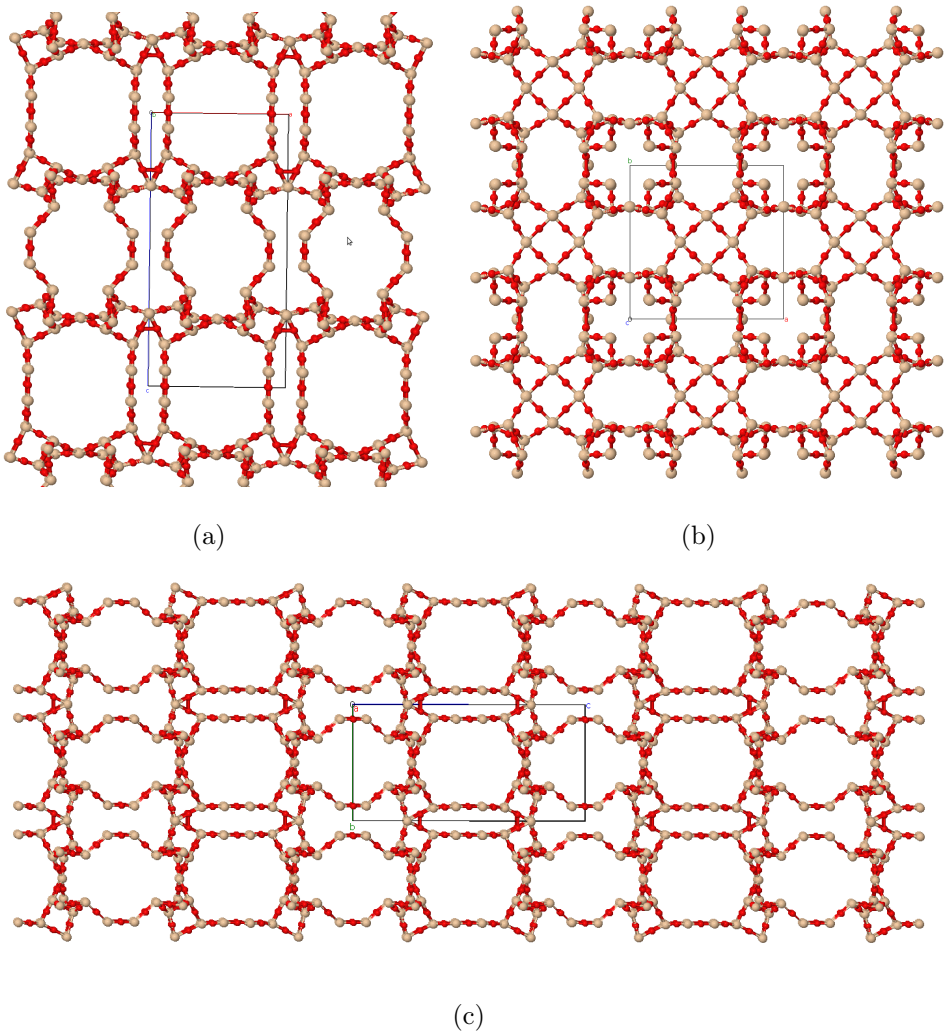
(b)



(c)

FIGURE 4.22 – Première topologie sélectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **ITE** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **ISV**, la figure 4.23 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement OPX.

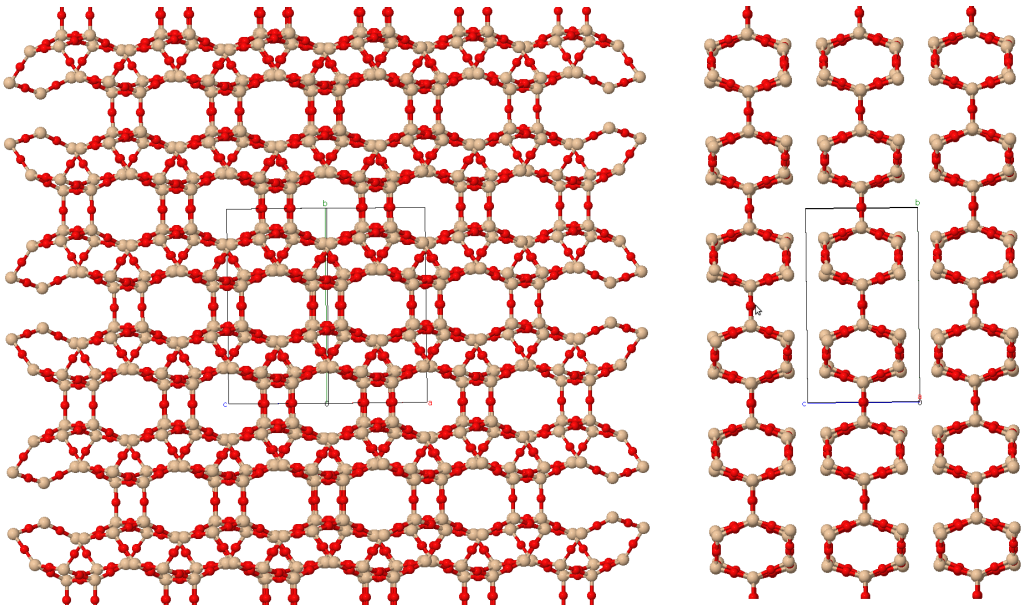


(a) (b)

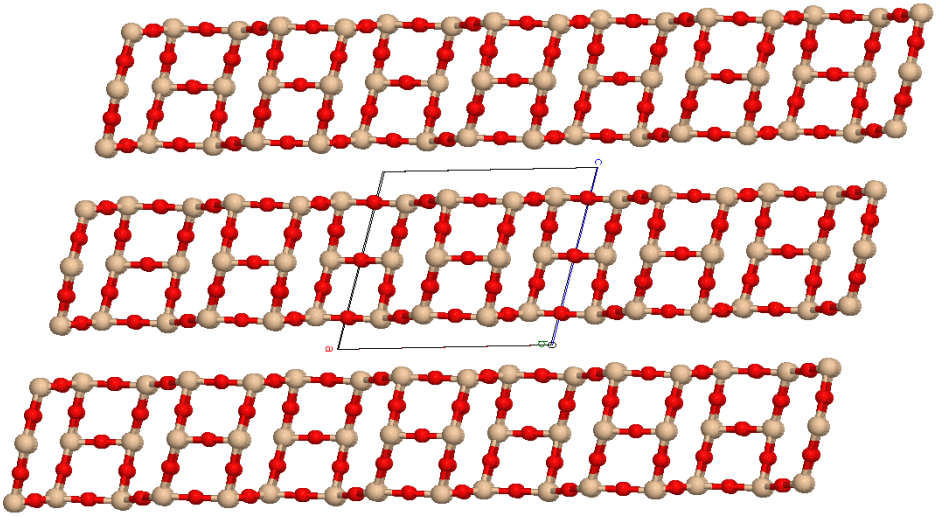
(c)

FIGURE 4.23 – Deuxième topologie sélectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **ISV** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **ITW**, la figure 4.24 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement OPX.



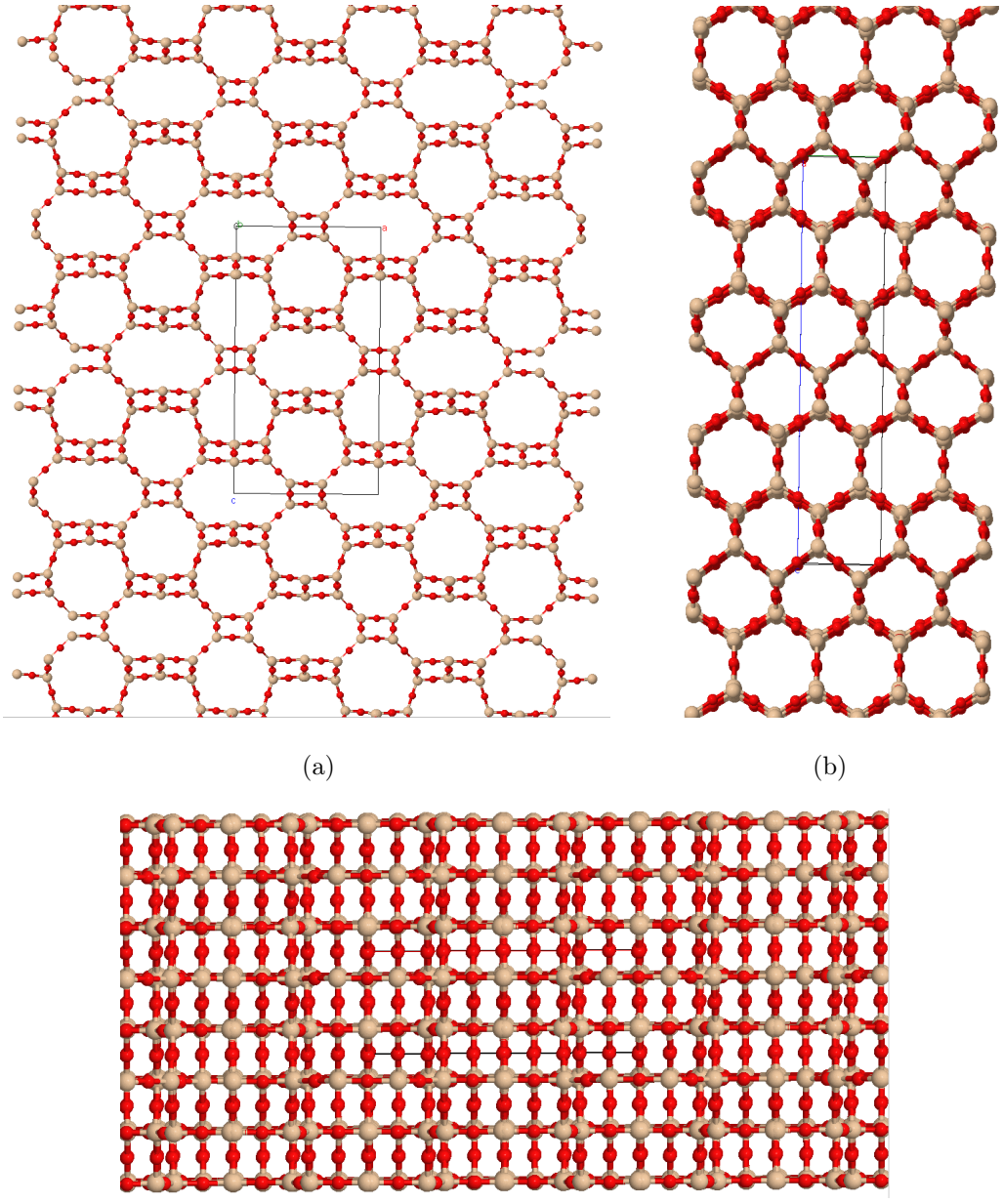
(a) (b)



(c)

FIGURE 4.24 – Troisième topologie selectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **ITW** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **CFI**, la figure 4.25 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement RX.

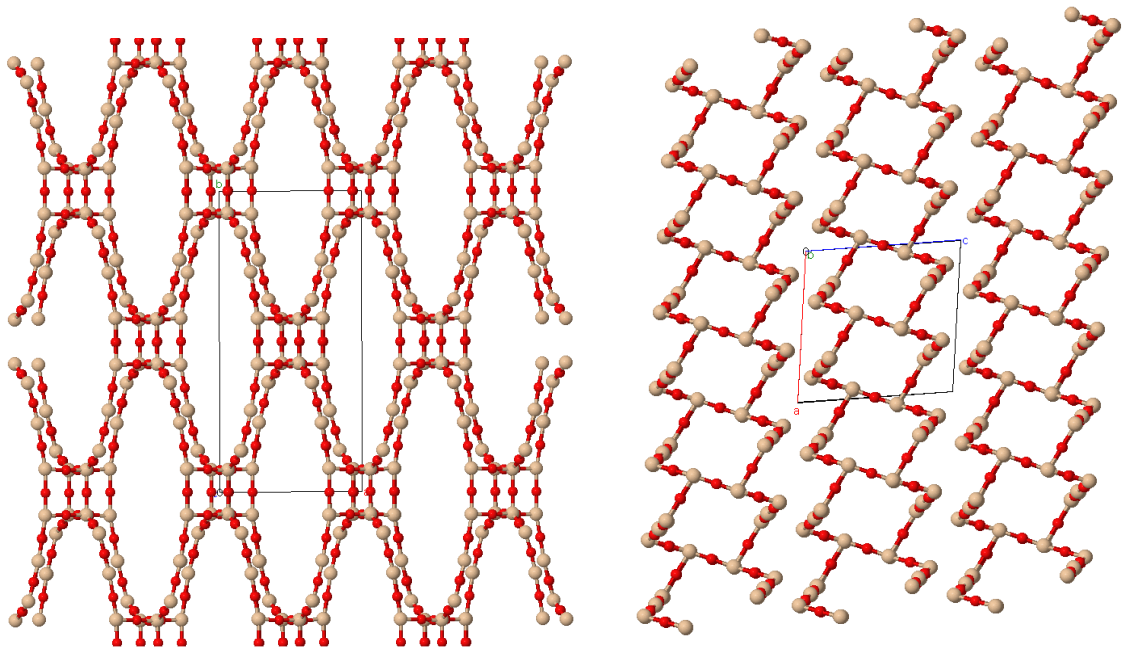


(a) (b)

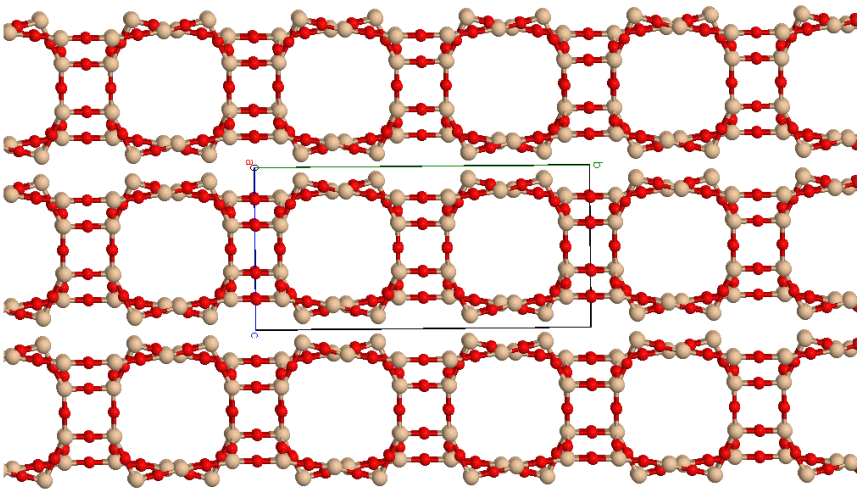
(c)

FIGURE 4.25 – Quatrième topologie sélectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **CFI** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **RTH**, la figure 4.26 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement UX.



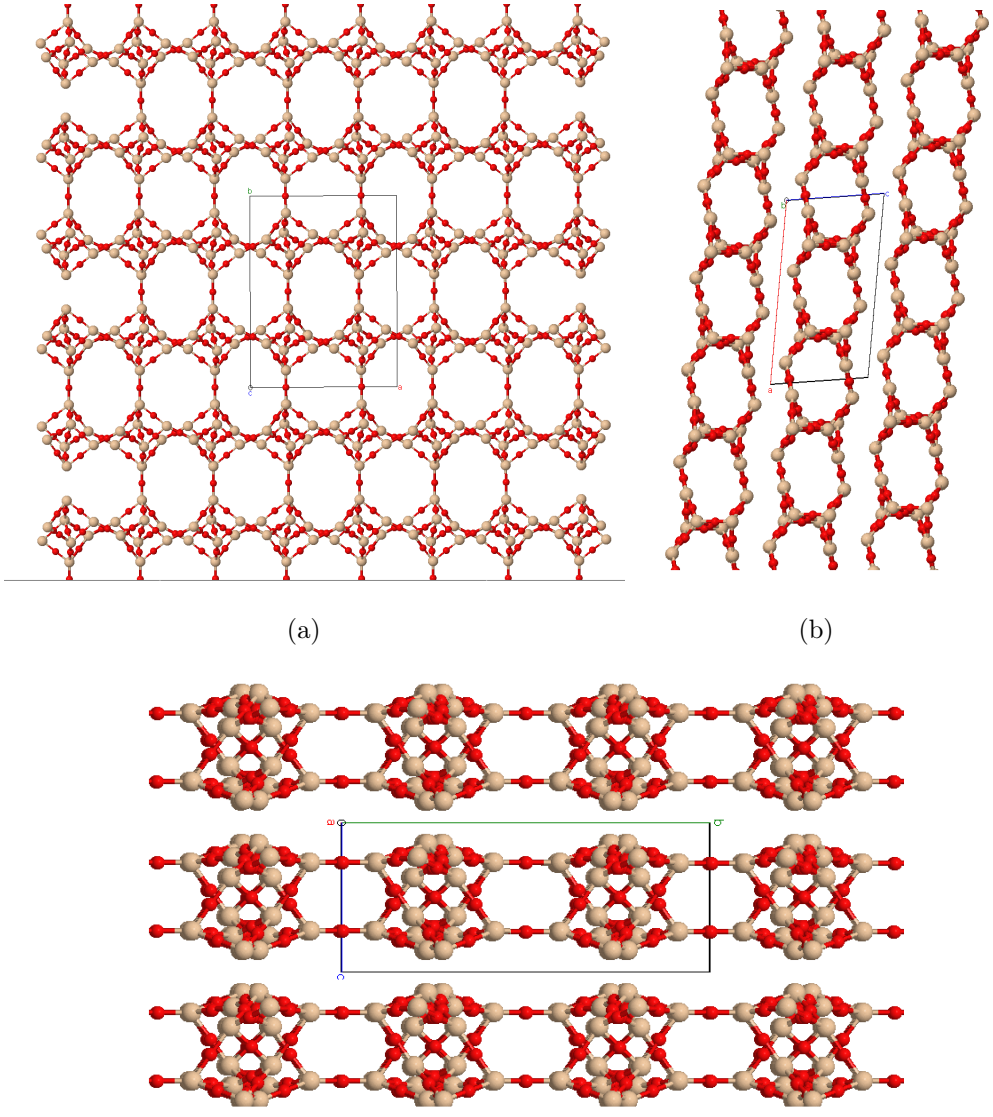
(a) (b)



(c)

FIGURE 4.26 – Cinquième topologie sélectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **RTH** avant la minimisation (UFF) et avant l'optimisation (BS).

À partir des paramètres de la topologie **STF**, la figure 4.27 montre une structure trouvée par le MEGA-HZ distribué avec l'opérateur de croisement UX.



(a) (b)

(c)

FIGURE 4.27 – Sixième topologie sélectionnée avec MEGA-HZ et la formulation 2 à partir des paramètres de **STF** avant la minimisation (UFF) et avant l'optimisation (BS).

4.5 Discussion

L'objectif de ce chapitre est de traiter le problème de la résolution structurale des zéolithes. D'un point de vue algorithmique, la génération de nouvelles topologies stables de zéolithes inconnues est un défi très intéressant pour aider les chimistes dans la synthèse de nouveaux matériaux zéolitiques. Dans le cadre de ce travail, nous avons proposé deux formulations pour une fonction objectif qui évalue le potentiel des topologies générées. Ces formulations sont basées sur les pénalisations et les récompenses ce qui donne une grande liberté à l'algorithme pour trouver de nouvelles topologies. Deux variantes d'un AG sont proposées : le P-GHAZ et le MEGA-HZ. Pour la validation numérique de la formulation 1, les deux algorithmes arrivent à atteindre l'optimalité en moyenne dans 50% des tentatives avec un avantage pour le P-GHAZ. Pour la formulation 2, la validation numérique du MEGA-HZ distribué est aux alentours de 56%. Globalement, nous sélectionnons 16 à 20% des topologies pour l'analyse dans le laboratoire de chimie. Pour l'algorithme P-GHAZ, six topologies stables sont générées et pour le MEGA-HZ les résultats sont en cours de traitement dans le laboratoire de chimie.

Certaines propriétés des topologies zéolitiques, comme les distances entre les atomes T, sont connues (environ 3 Å pour les distances). Nous utilisons une relaxation pour prendre en compte ces propriétés. Nos expérimentations révèlent que plus la pénalisation est agressive, moins nos approches proposent de nouvelles topologies. Cependant, cette relaxation peut conduire à de nombreuses solutions optimales numériquement mais avec une valeur énergétique élevée, ce qui signifie qu'elles ne peuvent pas être synthétisées. Les résultats obtenus confirment que les formulations de notre fonction objectif ont la capacité de proposer des topologies stables.

À travers ce problème, plusieurs perspectives s'offrent à nous. Sur le court terme, nous pouvons proposer d'autres pénalisations ou récompenses que les chimistes jugent intéressantes pour améliorer la qualité des topologies générées. Une autre perspective est l'orientation des algorithmes vers des topologies avec des caractéristiques spécifiques.

Nous pouvons également implémenter et repenser les niveaux de parallélisation définis dans cette thèse pour réduire le temps d'exécution et améliorer les résultats. D'autres métaheuristiques peuvent être proposées pour ce problème afin de voir le type de structures qu'elles peuvent générer. Sur le long terme, nous pouvons développer un logiciel qui permet de générer de nouvelles topologies hypothétiques. En effet, après le travail effectué par notre algorithme, les chimistes utilisent d'autres logiciels pour minimiser les structures, ce qui implique l'existence d'autres problèmes d'optimisation. D'autres traitements tels que le calcul d'énergie des structures sont réalisés. L'idée serait de proposer un logiciel qui génère, minimise et calcule le potentiel des zéolithes, et qui est capable de comparer la topologie générée aux autres topologies connues pour déterminer s'il s'agit d'une nouvelle topologie. Un tel logiciel peut nécessiter différentes compétences en informatique et dans le domaine des zéolithes.

Conclusion

Dans ce chapitre, deux variantes d'AG qui utilisent deux niveaux d'hybridation et deux niveaux de parallélisation pour résoudre le problème de la résolution structurale des zéolithes ont été proposées. Elles utilisent deux formulations de la fonction objectif pour évaluer le potentiel des topologies générées. Des expérimentations sur chaque formulation ont été effectuées pour évaluer le potentiel de nos algorithmes à résoudre ce problème.

Conclusion et Perspectives

La parallélisation et l'hybridation des métaheuristiques sont des moyens efficaces pour résoudre les problèmes d'optimisation. Durant ce travail doctoral, nous avons élaboré des algorithmes utilisant différents niveaux de parallélisation et d'hybridation. Ces derniers ont été mis en place essentiellement dans l'objectif d'améliorer la qualité des solutions trouvées. Leur efficacité a pu être observée dans la résolution de plusieurs problèmes d'optimisation difficile. Cependant, il ne suffit pas de paralléliser et d'hybrider une métaheuristique, il faut aussi le faire d'une façon adaptée au problème sur lequel la métaheuristique est utilisée. À travers trois problèmes d'optimisation, le comportement de la parallélisation et de l'hybridation a été expérimenté. L'objectif étant d'analyser l'impact des différents niveaux définis dans la résolution des problèmes expérimentés.

Le premier chapitre de ce document a présenté le contexte et l'état de l'art de notre travail sur les métaheuristiques hybrides et parallèles. Les différentes connaissances de base qu'il faut avoir sur les architectures parallèles et les techniques d'hybridation et de parallélisation des métaheuristiques ont été présentées.

La deuxième partie de cette thèse a présenté les contributions réalisées. Trois problèmes d'optimisation ont été testés pour ce travail. Dans le deuxième chapitre, une première contribution a été proposée. Il s'agit d'un algorithme hybride et parallèle d'optimisation par colonies de fourmis, appliqué au problème du voyageur de commerce. Dans cet algorithme, cinq niveaux de parallélisation et deux niveaux d'hybridation sont utilisés. En se basant sur la littérature, les niveaux de parallélisation proposés sont : la parallélisa-

tion des algorithmes, la parallélisation des structures de voisinage des S-métaheuristiques, la parallélisation au niveau de la solution, la parallélisation des individus pour les P-métaheuristiques et la parallélisation des données. Deux niveaux d'hybridation ont été également proposés qui sont : l'hybridation à bas niveau et l'hybridation à haut niveau.

Le troisième chapitre concerne le problème d'affectation quadratique. Quatre variantes basées sur une recherche taboue itérative avec des diversifications adaptatives ont été proposées. Les quatre variantes se distinguent par les protocoles d'échange d'informations entre les processus parallèles et les méthodes de diversification. Ces diversifications sont : la diversification de Glover, le croisement uniforme, la perturbation et la relocalisation. La recherche taboue itérative est une recherche taboue avec une hybridation à haut niveau et les diversifications adaptatives représentent une hybridation à bas niveau. Pour la parallélisation, le niveau de parallélisation des algorithmes a été utilisé en un premier lieu. Ce niveau parallélise les différentes recherches taboues itératives avec différentes solutions de départ. Ensuite, le niveau de parallélisation des structures de voisinage a été ajouté sur l'évaluation des voisins dans chaque recherche taboue. Deux expérimentations ont été menées sur les quatre variantes. Le but de la première expérimentation est d'étudier : l'impact de l'échange des données dans l'amélioration de la qualité des solutions, l'impact du changement de la diversification entre la diversification de Glover et le croisement uniforme et l'impact de la méthode de coopération. La deuxième expérimentation a pour but de comparer nos variantes avec six travaux de la littérature. En ajoutant le niveau de parallélisation des structures de voisinage, d'autres expérimentations ont été effectuées sur les instances de grandes tailles.

Le dernier chapitre est consacré au problème de la résolution structurale des zéolithes. Deux nouvelles formulations de la fonction objectif ont été proposées pour évaluer le potentiel des structures zéolitiques trouvées. La première formulation est basée exclusivement sur les pénalisations. Ces pénalisations concernent la distance entre les atomes, les valeurs des angles, la connectivité de la structure, le nombre de liaisons des atomes

pour garantir la stabilité et la présence des 3MR. La deuxième formulation reprend les pénalisations de la première formulation et ajoute deux fonctions de récompense qui affinent le choix des structures. Ces récompenses concernent la formation des XMR et celle des 4MR. Ces éléments caractérisent la structure des zéolithes que nous voulons obtenir. Par la suite, deux algorithmes génétiques ont été proposés pour résoudre ce problème. La première proposition est un algorithme génétique qui utilise les éléments principaux de l'évolution artificielle. Pour cette proposition, le niveau de parallélisation des individus a été utilisé sur le GPU pour évaluer les structures générées par les individus de la population. Une hybridation à bas niveau a également été utilisée en remplaçant la mutation par une recherche locale. Cette métaheuristique est appliquée uniquement avec la première formulation. Notre deuxième proposition se base sur les concepts de mémorisation et de prédiction. Cette proposition a été appliquée, dans un premier temps, avec la première formulation en utilisant le niveau de parallélisation des individus. Par la suite, le niveau de parallélisation des algorithmes et l'hybridation à haut niveau ont été ajoutés en générant plusieurs populations qui évoluent en parallèle sur différents processus. Afin de valider les structures zéolitiques, deux types de validation ont été nécessaires : la validation numérique et la validation des structures zéolitiques. La validation numérique a consisté à relever les structures qui obtiennent l'optimalité par les algorithmes. Pour la validation des structures zéolitiques, elle a été faite en deux étapes. La première étape a été la validation visuelle que nous avons effectuée pour les topologies que l'algorithme a générées avec une valeur optimale. La deuxième étape a été effectuée dans le laboratoire de chimie pour le calcul de l'énergie et de la stabilité de la topologie. Au niveau de la validation numérique de la première formulation, nos deux algorithmes sont arrivés à atteindre l'optimalité en moyenne dans 50% des tentatives. Pour la deuxième formulation, la validation numérique du deuxième algorithme est aux alentours de 56%. Globalement, 16 à 20% des topologies ont été sélectionnées pour l'analyse dans le laboratoire de chimie. Pour le premier algorithme, six topologies stables ont été validées dont trois sont non répertoriées

dans les bases de données des zéolithes. Pour le deuxième algorithme, 47 topologies ont été sélectionnées. Elles sont en cours de traitement dans le laboratoire de chimie.

Cette thèse nous permet d'ouvrir plusieurs perspectives. En ce qui concerne les niveaux de parallélisation et d'hybridation, différentes possibilités d'expérimentation s'offrent à nous. Nous prenons par exemple, l'expérimentation d'autres topologies d'échange de données dans le niveau de parallélisation des algorithmes comme la topologie aléatoire. De plus, la coopération entre les processus pourra exploiter, dans certains cas, la parallélisation au niveau de la solution. Plusieurs autres techniques de parallèlisation et d'hybridation restent à tester et à analyser.

Par ailleurs, d'autres problèmes d'optimisation peuvent être abordés dans le cadre de la création d'une bibliothèque de résolution avec les métaheuristiques distribuées et massivement parallèles. Cela peut se faire en appliquant les méthodes développées dans cette thèse, ou bien en proposant de nouvelles métaheuristiques.

Le travail effectué sur chaque problème traité dans cette thèse peut également être poursuivi ou étendu. Pour le problème d'affectation quadratique, par exemple, la résolution des instances de grande taille telle que les instances de taille 343 ou encore de taille 729, disponibles sur *http ://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap. html* représente une perspective intéressante. D'un côté, la parallélisation nous permet de réduire le temps de calcul et de proposer des métaheuristiques robustes. D'un autre côté, l'hybridation permet de trouver le bon compromis entre l'intensification et la diversification pour résoudre ce type d'instance. Pour le problème de la résolution structurale des zéolithes, d'autres formulations de la fonction objectif sont possibles pour orienter la recherche vers un type particulier de zéolithes. L'utilisation des niveaux d'hybridation et de parallélisation du deuxième chapitre de cette thèse constitue également une perspective afin d'améliorer les algorithmes du chapitre quatre.

Bibliographie

Aarts, E., & Lenstra, J. K. (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons.

Abdelkafi, O., Idoumghar, L., & Lepagnot, J. (2015a). Comparison of two diversification methods to solve the quadratic assignment problem. *International Conference On Computational Science (ICCS 2015) : Computational Science at the Gates of Nature, Reykjavik, Iceland, June 1-3*, *51*, 2703–2707.

Abdelkafi, O., Idoumghar, L., & Lepagnot, J. (2015b). Distributed multistart hybrid iterative tabu search. *IEEE International Conference on Systems, Man, and Cybernetics, Kowloon, Hong kong, October 2015*, (pp. 1962–1967).

Abdelkafi, O., Idoumghar, L., & Lepagnot, J. (2016a). A survey on the metaheuristics applied to QAP for the graphics processing units. *Parallel Processing Letters*, *26*, 1–20.

Abdelkafi, O., Idoumghar, L., Lepagnot, J., & Brévilliers, M. (2015c). A GPU-based parallel neighborhood evaluation for ITSSD. *Proceedings of the 12th Biennal International Conference on Artificial Evolution, Lyon, France, October 2015*, (pp. 327–334).

Abdelkafi, O., Idoumghar, L., Lepagnot, J., & Brévilliers, M. (2016b). Data exchange topologies for the DISCO-HITS algorithm to solve the QAP. *International Conference on Swarm Intelligence Based Optimization, Mulhouse, France, June 2016*, (pp. 30–36).

Abdelkafi, O., Lepagnot, J., & Idoumghar, L. (2014). Multi-level parallelization for hybrid ACO. *Swarm Intelligence Based Optimization, Lecture Notes in Computer Science*, *8472*, 60–67.

André, R. B., Trond, R. H., Christian, S., & Geir, H. (2013). GPU computing in discrete optimization. Part 1 : introduction to the GPU. *EURO Journal on Transportation and Logistics*, *2*, 129–157.

Aouad, M. I., Idoumghar, L., Schott, R., & Zendra, O. (2010). Sequential and distributed hybrid GA-SA algorithms for energy optimization in embedded systems. *The IADIS International Conference Applied Computing, Timisoara, Romania, October 2010*, (pp. 167–174).

Archibald, B., Brümmer, O., Devenney, M., Giaquinta, D. M., Jandeleit, B., Weinberg, W. H., & Weskamp, T. (2005). Combinatorial aspects of materials science. In K. C. Nicolaou, R. Hanko, & W. Hartwig (Eds.), *Handbook of Combinatorial Chemistry : Drugs, Catalysts, Materials* (pp. 1017–1062). Wiley-VCH Verlag GmbH & Co. KGaA.

Argauer, R. J., & Landolt, G. R. (1972). Crystalline zeolite ZSM-5 and method of preparing the same.

Bäck, T. (1996a). *Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press Oxford, UK.

Bäck, T. (1996b). *Evolutionary algorithms in theory and practice evolution strategies, evolutionary programming, genetic algorithms*. New York : Oxford University Press. URL : `$http://public.eblib.com/choice/publicfullrecord.aspx?p=430643$`.

Baerlocher, C., & McCusker, L. B. (). Database of zeolite structures. URL : `$http://www.iza-structure.org/databases/$`.

Barrer, R. M. (1982). *Hydrothermal chemistry of zeolites*. London, New York : Academic Press.

Barrón, C., Gómez, S., Romero, D., & Saavedra, A. (1999). A genetic algorithm for lennard-jones atomic clusters. *Applied Mathematics Letters*, *12*, 85–90.

Baumes, L. A., Kruger, F., & Collet, P. (2013). High-performance computing for accelerated zeolitic materials modeling. In K. Rajan (Ed.), *Informatics for Materials Science and Engineering* (pp. 315–347). Butterworth-Heinemann, Oxford.

Baumes, L. A., Kruger, F., Jimenez, S., Collet, P., & Corma, A. (2011). Boosting theoretical zeolitic framework generation for the determination of new materials structures using gpu programming. *Physical Chemistry Chemical Physics*, *13*, 4674–4678.

Benlic, U., & Hao, J. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, *42*, 584–595.

Benlic, U., & Hao, J. K. (2013). Breakout local search for the quadratic assignement problem. *Applied Mathematics and Computation*, *219*, 4800–4815.

Bhaba, R. S., Wilbert, E. W., & Gary, L. H. (1998). Locating sets of identical machines in a linear layout. *Annals of Operations Research*, *77*, 183–207.

Bland, R. G., & Shallcross, D. F. (1989). Large traveling salesman problems arising experiments in X-ray crystallography : A preliminary report on computation. *Operations Research Letters*, *8*, 125–128.

Blatov, V. A., Shevchenko, A. P., & Proserpio, D. M. (2014). Applied topological analysis of crystal structures with the program package topospro. *Crystal Growth and Design*, *14*, 3576–3586.

Blum, C., Aguilera, M. B., Roli, A., & Sampels, M. (2008). *Hybrid Metaheuristics - An Emerging Approach to Optimization* volume 114. Studies in Computational Intelligence, Springer.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Computing Surveys*, *35*, 268–308.

Bluma, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization : A survey. *Applied Soft Computing*, *11*, 4135–4151.

Brévilliers, M., Abdelkafi, O., Lepagnot, J., & Idoumghar, L. (2015). Idol-Guided backtracking search optimization algorithm. *Proceedings of the 12th Biennal International Conference on Artificial Evolution, Lyon, France, October 2015*, (pp. 277–284).

Brévilliers, M., Abdelkafi, O., Lepagnot, J., & Idoumghar, L. (2016). Fast hybrid BSA-DE-SA algorithm on GPU. *International Conference on Swarm Intelligence Based Optimization, Mulhouse, France, June 2016*, (pp. 46–53).

Burkard, R. E., Karisch, S. E., & Rendl, F. (1997). QAPLIB - A quadratic assignment problem library. *Journal of Global Optimization*, *10*, 391–403.

Bushuev, Y. G., & Sastre, G. (2009). Atomistic simulations of structural defects and water occluded in ssz-74 zeolite. *The Journal of Physical Chemistry C*, *113*, 10877–10886.

Cáceres, E. N., Fingler, H., Mongelli, H., & Song, S. W. (2012). Ant colony system based solutions to the quadratic assignment problem on GPGPU. *41st International Conference on Parallel Processing Workshops, Pittsburgh, PA, USA, September 2012*, *1*, 314–322.

Cahon, S., Melab, N., & Talbi, E. G. (2004). ParadisEO : A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, *10*, 357–380.

Cecilia, J. M., Garcia, J. M., Nisbet, A., Amos, M., & Ujaldon, M. (2013). Enhancing data parallelism for ant colony optimization on GPUs. *Journal of Parallel and Distributed Computing*, *73*, 42–51.

Cerius^2 (2000). Molecular simulations inc.

Chaparala, A., Novoa, C., & Qasem, A. (2014). A SIMD solution for the quadratic assignment problem with GPU acceleration. *Annual Conference on Extreme Science and Engineering Discovery Environment, Atlanta, GA, USA, July 2014* , (pp. 1–8).

Chen, S.-M., & Chien, C.-Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, *38*, 14439–14450.

Chirico, U. (2004). A java framework for ant colony systems. *Siemens Informatica*, .

Cochrane, E. M., & Beasley, J. E. (2003). The co-adaptive neural network approach to the euclidean traveling salesman problem. *Neural Networks*, *16*, 1499–1525.

Collet, P., Lutton, E., Schoenauer, M., & Louchet, J. (2000). Take it easea. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, & H. Schwefel (Eds.), *Parallel Problem Solving from Nature PPSN VI* chapter 87. (pp. 891–901). Springer Berlin Heidelberg volume 1917 of *Lecture Notes in Computer Science*.

Cotta, C., Aldana, J. F., Nebro, A. J., & Troya, J. M. (1995). Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In *Artificial Neural Nets and Genetic Algorithms* (pp. 277–280). Springer.

Cronstedt, A. F. (1756). Rön och beskrifning om en obekant bärg art som kallas zeolites. *Kungliga Vetenskapsakademiens Handlingar*, *17*, 120–123.

Czapinski, M. (2013). An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing*, *73*, 1461–1468.

Daven, D. M., Tit, N., Morris, J. R., & Ho, K. M. (1996). Structural optimization of lennard-jones clusters by a genetic algorithm. *Chemical Physics Letters*, *256*, 195–200.

Deem, M. W., & Newsam, J. M. (1989). Determination of 4-connected framework crystal structures by simulated annealing. *Nature*, *342*, 260–262.

Deem, M. W., Newsam, J. M., & John, M. (1992). Framework crystal structure solution by simulated annealing : test application to known zeolite structures. *Journal of the American Chemical Society*, *114*, 7189–7198.

Delévacq, A., Delisle, P., Gravel, M., & Krajecki, M. (2013). Parallel ant colony optimization on graphics processing units. *Journal of Parallel and Distributed Computing*, *73*, 52–61.

Delgado, O. F., Dress, A. W. M., Huson, D. H., Klinowski, J., & Mackay, A. L. (1999). Systematic enumeration of crystalline networks. *Nature*, *400*, 644–647.

Delgado-Friedrichs, O. (2003). Data structures and algorithms for tilings I. *Theoretical Computer Science*, *303*, 431–445.

Desrochers, M., & Laporte, G. (1991). Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Reasearch Letters*, *10*, 27–36.

Dorigo, M. (1992). *Optimization : learning and natural algorithms*. Ph.D. thesis Politecnico di Milano, Italy.

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

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA.

Drezner, Z. (2008). Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers and Operations Research*, *35*, 717–736.

Drezner, Z., Hahn, P. M., & Taillard, E. (2005). Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations research*, *139*, 65–94.

Duman, E., & Or, I. (2007). The quadratic assignement problem in the context of the printed circuit board assembly process. *Computers and Operations Research*, *34*, 163–179.

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms* (pp. 187–202). San Mateo, California : Morgan Kaufmann Publishers.

Falcioni, M., & Deem, M. W. (1999). A biased monte carlo scheme for zeolite structure solution. *Journal of Chemical Physics*, *110*, 1754–1766.

Fan, E. (2002). *Global optimization of the Lennard-Jones atomic clusters*. Ph.D. thesis McMaster University.

Flynn, M. J. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, *54*, 1901–1909.

Foster, M. D., & Treacy, M. M. J. (). A database of hypothetical zeolite structures. URL : `$http://www.hypotheticalzeolites.net$`.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, *32*, 675–701.

Gale, J. (1997). Gulp : A computer program for the symmetry-adapted simulation of solids. *Journal of the Chemical Society, Faraday Transactions*, *93*, 629–637.

Glover, F. (1990). Tabu search : A tutorial. *Center for Applied Artificial Intelligence, University of Colorado, Boulder, Colorado 80309-0419*, (pp. 74–94).

Glover, F. (1998). A template for scatter search and path relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Artificial Evolution, Lecture Notes in Computer Science* (pp. 13–54). Springer volume 1363.

Gong, D., & Ruan, X. (2004). A hybrid approach of GA and ACO for TSP. *IEEE Fifth World Congress on Intelligent Control and Automation, June 2004*, *3*, 2068–2072.

Grosse-Kunstleve, R. W., McCusker, L. B., & Baerlocher, C. (1997). Powder diffraction data and crystal chemical information combined in an automated structure determination procedure for zeolites. *Journal of Applied Crystallography*, *30*, 985–995.

Grosse-Kunstleve, R. W., McCusker, L. B., & Baerlocher, C. (1999). Zeolite structure determination from powder diffraction data : applications of the FOCUS method. *Journal of Applied Crystallography*, *32*, 536–542.

Guisnet, M., & Gilson, J. (2002). Introduction to zeolite science and technology. In M. Guisnet, & J. Gilson (Eds.), *Zeolites for Cleaner Technologies* Catalytics Science Series chapter 1. (pp. 1–28). Imperial College Press.

Gutin, G., & Punnen, A. P. (2002). *The Traveling Salesman Problem and Its Variations*. Springer.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MIT : University of Michigan Press.

Honvault, P. (2004). Trigonométrie. In *Une approche possible de la géométrie plane* chapter 4. (p. 41). URL : `$https://books.google.fr/books?id=t43c0p50NkQC&pg=PA41&redir_esc=y#v=onepage&q&f=false$`.

Hulianytskyi, L., & Sirenko, S. (2010). Cooperative model-based metaheuristics. *Electronic Notes in Discrete Mathematics*, *36*, 33–40.

Hussin, M. S., & Stützle, T. (2009). *Hierarchical Iterated Local Search for the Quadratic Assignment Problem*. Technical Report IRIDIA - Technical Report Series.

Hussin, M. S., & Stützle, T. (2014). Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Computers and Operations Research*, *43*, 286–291.

Idoumghar, L., Chérin, N., Siarry, P., Roche, R., & Miraoui, A. (2013). Hybrid ICA-PSO algorithm for continuous optimization. *Applied Mathematics and Computation*, *219*, 11149–11170.

Idoumghar, L., Melkemi, M., Schott, R., & Aouad, M. I. (2011). Hybrid PSO-SA type algorithms for multimodal function optimization and reducing energy consumption in embedded systems. *Applied Computational Intelligence and Soft Computing*, *2011*, 1–12.

Idoumghar, L., & Schott, R. (2009). Two distributed algorithms for the frequency assignment problem in the field of radio broadcasting. *IEEE Transactions on Broadcasting, Romania, Mars 2009*, *55*, 223–229.

Idoumghar, L., Schott, R., & Alabau, M. (2001). New hybrid genetic algorithms for the frequency assignment problem. *13th International IEEE Conference on Tool with Artificial Intelligence, Dallas-Texas, USA, Novembre 2001*, (pp. 136–142).

James, T., Rego, C., & Glover, F. (2009a). A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, *195*, 810–826.

James, T., Rego, C., & Glover, F. (2009b). Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems Man And Cybernetics-Part A : Systems and Humans*, *39*, 579–596.

Jones, G., Willett, P., & Glen, R. C. (1995). Molecular recognition of receptor sites using a genetic algorithm with a description of desolvation. *Journal of Molecular Biology*, *245*, 43–53.

Kadluczka, P., & Wala, K. (1995). Tabu search and genetic algorithms for the generalized graph partitioning problem. *Control Cybernetics*, *24*, 459–476.

Kim, J. S. (2002). Distributed genetic algorithm with multiple populations using multi-agent. In *High Performance Computing* (pp. 329–334). Springer.

Kirk, D. B., & Hwu, W. M. (2010). *Programming massively parallel processors : a hands-on approach*. Morgan kaufmann, 280 pages.

Kokotailo, G. T., Lawton, S. L., Olson, D. H., & Meier, W. M. (1978). Structure of synthetic zeolite zsm-5. *Nature*, *272*, 437–438.

Koopmans, T., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*, 53–76.

Krasnogor, N., & Smith, J. (2005). A tutorial for competent memetic algorithms : model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, *9*, 474–488.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*, 497–520.

Laporte, G. (1992). The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*, 345–358.

Lepagnot, J., Idoumghar, L., & Fodorean, D. (2013). Hybrid imperialist competitive algorithm with simplex approach : Application to electric motor design. *IEEE International Conference on Systems Man and Cybernetics, Manchester, United Kingdom, October 2013*, (pp. 2454–2459).

Lewis, D. W., Sankar, G., Wyles, J. K., & Thomas, J. M. (1997). Synthesis of a small-pore microporous material using a computationally designed template. *Angewandte Chemie, International Edition in English*, *36*, 2675–2677.

Li, J., Corma, A., & Yu, J. (2015). Synthesis of new zeolite structures. *Chemical Society Reviews*, *44*, 7112–7127.

Loiola, E. M., de Abreu, N. M. M., Netto, P. O. B., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, *176*, 657–690.

Lozano, M., & Garcia-Martinez, C. (2010). Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification : overview and progress report. *Computers and Operations Research*, *37*, 481–497.

Luong, T. V. (2011). *Métaheuristique parallèles sur GPU*. Ph.D. thesis Ecole doctorale sciences Pour l'ingenieur Universite Lille Nord-de-France.

Luong, T. V., Melab, N., & Talbi, E.-G. (2010a). Algorithmes évolutionnaires parallèles sur gpu. *MajecSTIC'10*, (pp. 1–9).

Luong, T. V., Melab, N., & Talbi, E. G. (2010b). Parallel hybrid evolutionary algorithms on GPU. *IEEE Congress on Evolutionary Computation, Barcelona, Spain, July 2010*, (pp. 1–8).

Luong, T. V., Melab, N., & Talbi, E. G. (2013). GPU computing for parallel local search metaheuristic algorithms. *IEEE Transactions on Computers*, *62*, 173–185.

Luong, T. V., & Taillard, E. (2012). *GPU-based Approaches for Hybrid Metaheuristics*. Technical Report CPUGPU Project HES-SO RCSO-TIC, University of Applied Sciences of Western Switzerland, Yverdon-Les-Bains, Switzerland.

Masutti, T. A. S., & Castro, L. N. D. (2009). A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. *Information Sciences*, *179*, 1454–1468.

McCusker, L. B., Liebau, F., & Engelhardt, G. (2001). Nomenclature of structural and compositional characteristics of ordered microporous and mesoporous materials with inorganic hosts(iupac recommendations 2001). *Pure and Applied Chemistry*, *73*, 381–394.

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, *4*, 337–352.

Misevicius, A. (2004). An improved hybrid genetic algorithm : New results for the quadratic assignment problem. *Knowledge Based Systems*, *17*, 65–73.

Misevicius, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, *30*, 95–111.

Misevicius, A., & Kilda, B. (2006). Iterated tabu search : An improvement to standard tabu search. *Information Technology and Control*, *35*, 187–197.

Paillaud, J. L., Harbuzaru, B., Patarin, J., & Bats, N. (2004). Extra-large-pore zeolites with two-dimensional channels formed by 14 and 12 rings. *Science*, *304*, 990–992.

Polak, G. G. (2005). On a special case of the quadratic assignment problem with an application to storage-and-retrieval devices. *Annals of Operations Research*, *138*, 223–233.

Pophale, R., Daeyaert, F., & Deem, M. (2013). Computational prediction of chemically synthesizable organic structure directing agents for zeolites. *Journal of Materials Chemistry A*, *1*, 6750–6760.

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. In J. Mira, & J. R. Álvarez (Eds.), *Artificial Intelligence and Knowledge Engineering Applications : A Bioinspired Approach, Lecture Notes in Computer Science* (pp. 41–53). Springer volume 3562.

Puris, A., Bello, R., & Herrera, F. (2010). Analysis of the efficacy of a two-stage methodology for ant colony optimization : Case of study with tsp and qap. *Expert Systems with Applications*, *37*, 5443–5453.

Raidl, G. R., Puchinger, J., & Blum, C. (2010). Metaheuristic hybrids. In M. Gendreau, & J. Y. Potvin (Eds.), *Handbook of Metaheuristics, International Series in Operations Research & Management Science* (pp. 469–496). Springer volume 146.

Rappe, A. K., Casewit, C. J., Colwell, K. S., Goddard, W. A., & Skiff, W. M. (1992). Uff, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American Chemical Society*, *114*, 10024–10035.

Reinelt, G. (1991). TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, *3*, 376–384.

Roberge, V., Tarbouchi, M., & Okou, F. (2015). gpuMF : A framework for parallel hybrid metaheuristics on GPU with application to the minimization of harmonics in multilevel inverters. *International Journal of Process Systems Engineering*, *3*, 20–41.

Schmidt, J. E., Deem, M. W., & Davis, M. E. (2014). Synthesis of a specified, silica molecular sieve by using computationally predicted organic structure-directing agents. *Angewandte Chemie International Edition*, *53*, 8372–8374.

Schmitt, K. D., & Kennedy, G. (1994). Toward the rational design of zeolite synthesis : The synthesis of zeolite zsm-18. *Zeolites*, *14*, 635–642.

Siarry, P. (2014). *Métaheuristiques*. Eyrolles, 516 pages.

Snir, M., Otto, S., Huss-Lederman, S., Walter, D., & Dongarra, J. (1996). *MPI : The Complete Reference*. MIT Press Boston.

Somhom, S., Modares, A., & Enkawa, T. (1997). A self-organizing model for the traveling salesman problem. *Journal of the Operational Research Society*, *48*, 919–928.

Speybroeck, V. V., Hemelsoet, K., Joos, L., Waroquier, M., Bell, R., & Catlow, C. (2015). Advances in theory and their application within the field of zeolite chemistry. *Chemical Society Reviews*, *44*, 7044–7111.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, *174*, 1519–1539.

Stützle, T., & Hoos, H. (1997). MAX-MIN ant system and local search for the traveling salesman problem. *IEEE International Conference on Evolutionary Computation, Indianapolis, USA, April 1997*, (pp. 309–314).

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, *37*, 1899–1911.

Taillard, E. (1991). Robust taboo search for the quadratic assignement problem. *Parallel Computing*, *17*, 443–455.

Talbi, E. G. (2009). *Metaheuristics : from design to implementation*. John Wiley and Sons, 624 pages.

Talbi, E. G., Roux, O., Fonlupt, C., & Robillard, D. (2001). Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, *17*, 441–449.

Teghem, J., & Pirlot, M. (2002). *Optimisation approchée en recherche opérationnelle*. LAVOISIER, 97 pages.

Tosun, U. (2014). A new recombination operator for the genetic algorithm solution of the quadratic assignment problem. *Procedia Computer Science*, *32*, 29–36.

Tosun, U. (2015). On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, *39*, 267–278.

Treacy, M. M. J., Rivin, I., Balkovsky, E., Randall, K. H., & Foster, M. D. (2004). Enumeration of periodic tetrahedral frameworks. II. polynodal graphs. *Microporous Mesoporous Materials*, *74*, 121–132.

Tseng, L. Y., & Liang, S. C. (2006). A hybrid metaheuristic for the quadratic assignment problem. *Computational Optimization and Applications*, *34*, 85–113.

Tsutsui, S. (2006). cAS : ant colony optimization with cunning ants. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, & X. Yao (Eds.), *Parallel Problem Solving from Nature, Lecture Notes in Computer Science* (pp. 162–171). Springer volume 4193.

Tsutsui, S., & Fujimoto, N. (2009). Solving quadratic assignment problems by genetic algorithms with GPU computation : a case study. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation, Montreal, Canada, July 2009*, (pp. 2523–2530).

Tsutsui, S., & Fujimoto, N. (2011). ACO with tabu search on a GPU for solving QAPs using move-cost adjusted thread assignment. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, July 2011*, *1*, 1547–1554.

Ulutas, B. H., & Konak, S. K. (2012). An artificial immune system based algorithm to solve unequal area facility layout problem. *Expert Systems with Applications*, *39*, 5384–5395.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions On Evolutionary Computation*, *1*, 67–82.

Wu, Q., & Hao, J. K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, *42*, 355–365.

Zhang, Q., Sun, J., & Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, *9*, 192–200.

Zhua, W., Currya, J., & Marqueza, A. (2010). SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration. *International Journal of Production Research*, *48*, 1035–1047.

Résumé

De nombreux problèmes d'optimisation propres à différents secteurs industriels et académiques (énergie, chimie, transport, etc.) nécessitent de concevoir des méthodes de plus en plus efficaces pour les résoudre. Afin de répondre à ces besoins, l'objectif de cette thèse est de développer une bibliothèque composée de plusieurs métaheuristiques hybrides distribuées et massivement parallèles. Dans un premier temps, nous avons étudié le problème du voyageur de commerce et sa résolution par la méthode colonie de fourmis afin de mettre en place les techniques d'hybridation et de parallélisation. Ensuite, deux autres problèmes d'optimisation ont été traités, à savoir, le problème d'affectation quadratique (QAP) et le problème de la résolution structurale des zéolithes (ZSP). Pour le QAP, plusieurs variantes basées sur une recherche taboue itérative avec des diversifications adaptatives ont été proposées. Le but de ces propositions est d'étudier l'impact de : l'échange des données, des stratégies de diversification et des méthodes de coopération. Notre meilleure variante est comparée à six des meilleurs travaux de la littérature. En ce qui concerne le ZSP, deux nouvelles formulations de la fonction objective sont proposées pour évaluer le potentiel des structures zéolitiques trouvées. Ces formulations sont basées sur le principe de pénalisation et de récompense. Deux algorithmes génétiques hybrides et parallèles sont proposés pour générer des structures zéolitiques stables. Nos algorithmes ont généré actuellement six topologies stables, parmi lesquelles trois ne sont pas répertoriées sur le site Web du SC-IZA ou dans l'Atlas of Prospective Zeolite Structures.

Mots-clés: Métaheuristique, Hybridation, Algorithmes parallèles, GPU, MPI, Problème du voyageur de commerce, Problème d'affectation quadratique, Problème de résolution structurale des zéolithes.

Abstract

Many optimization problems specific to different industrial and academic sectors (energy, chemicals, transportation, etc.) require the development of more effective methods in resolving. To meet these needs, the aim of this thesis is to develop a library of several hybrid metaheuristics distributed and massively parallel. First, we studied the traveling salesman problem and its resolution by the ant colony method to establish hybridization and parallelization techniques. Two other optimization problems have been dealt, which are, the quadratic assignment problem (QAP) and the zeolite structure problem (ZSP). For the QAP, several variants based on an iterative tabu search with adaptive diversification have been proposed. The aim of these proposals is to study the impact of: the data exchange, the diversification strategies and the methods of cooperation. Our best variant is compared with six from the leading works of the literature. For the ZSP two new formulations of the objective function are proposed to evaluate the potential of the zeolites structures founded. These formulations are based on reward and penalty evaluation. Two hybrid and parallel genetic algorithms are proposed to generate stable zeolites structures. Our algorithms have now generated six stable topologies, three of them are not listed in the SC-IZA website or in the Atlas of Prospective Zeolite Structures.

Keywords: metaheuristic, hybridization, parallel algorithms, GPU, MPI, traveling salesman problem, quadratic assignment problem, zeolites structure problem.