République Algérienne Démocratique et Populaire
Ministère de l 'Enseignement Supérieur et de la Recherche Scientifique
UNIVERSITÉ AMAR TELIDJI-LAGHOUAT-
Faculté des Sciences

FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

Mémoire de MASTER

Domaine : Mathématiques et Informatique
Filière : Informatique
Option : Systémes d'Information et de Décision

Présenté par :

TRICHE ZAHIRA

THÈME

Optimisation des requêtes distribués avec HBMO

Noms du membre de jury :

M.LARADJ Chellama	Encadreur
M.YOUNES Guellouma	Examinateur
M.BADRA Kerrouche	président

N° d' ordre : ...../Année Universitaire2017/2018

Dédicaces

Je dédie ce modeste travail à :

Mes parents pour tout ce qu'il sont fait pour moi ;

Mes frères et mes soeurs ;

Tous ces ceux qui me sont chers.

ZAHIRA

Remerciement

Nous remercions **ALLAH** le tout puissant, maître des cieux et de la terre, qui nous a éclairé le chemin et nous à permis de mener à bien ce travail.

Tout d'abord nous tenons surtout à adresser nos plus vifs remerciements à monsieur **Chellama Laaredj**, qui nous a accepter de travailler sous sa direction. Nous ne saurons jamais oublier ses conseils judicieux.

Un merci pour tous nos enseignants .

Un grand merci à toutes les personnes qui nous ont soutenues de près ou de loin au cours de la réalisation de ce modeste travail.

Nos remercie à tous les membres du jury qui ont accepté de juger notre travail.

Resumé

Le traitement d'une requête distribuée implique l'accès aux données de plusieurs sites. Le coût de communication entre les sites, étant le coût dominant, doit être réduit afin d'améliorer le temps de réponse de la requête. Cela nécessite a l'optimiseur de requête de concevoir des nouvelles stratégie de traitement de requête distribuée. qui, pour une requête distribuée donnée,l'optimiseur de requête génère des plans de requête impliquant moins de nombre de sites afin de réduire le coût de communication entre eux. Dans ce mémoire, un algorithme de génération de plan de requête distribuée basée sur la l'algorithme HBMO (Honey Beee Mating Optimization), génère des requêtes pour une requête distribuée impliquant moins de sites et une concentration de relation plus élevée dans les sites participants.

Mots clés : optimisation, plan de requête distribué, l'algorithme HBMO.

ملخص

تتضمن معالجة الاستعلام موزع الوصول إلى البيانات من مواقع متعددة. يجب تقليل تكلفة الاتصال بين المواقع ، باعتبارها التكلفة المهيمنة ، من أجل تحسين وقت الاستجابة للطلب. سيتطلب ذلك مُحسِّن طلبات البحث لتصميم إستراتيجية معالجة استعلام موزّعة ، من أجل استعلام موزع معيّن ، سيؤدي إلى إنشاء خطط استعلام تتضمن عددًا أقل من المواقع لتقليل تكلفة الاتصال بين المواقع. في هذا البحث، تقوم خوارزمية توليد خطة التوزيع الموزعة على خوارزمية توليد خطة التوزيع الموزعة على خوارزمية تكاثر النحل
بإنشاء استعلامات للاستعلام الموزع الذي يتضمن عددًا أقل من المواقع وتركيز أعلى للعلاقة في المواقع المشاركة.

الكلمات المفتاحية :

تحسين؛ خطة الاستعلام الموزعة؛ الأدلة العليا؛ خوارزمية تكاثر النحل.

Abstract

le Processing a distributed query entails accessing data from multiple sites. The inter site communication cost, being the dominant cost, needs to be reduced in order to improve the query response time. This would require the query optimizer to devise a distributed query processing strategy that would, for a given distributed query, generate query plans involving fewer number of sites in order to reduce the inter site communication cost. In this document, a distributed query plan generation algorithm, based on the honey bee mating optimization (HBMO) algorithm that generates query plans for a distributed query involving less number of sites and higher relation concentration in the participating sites.

Keywords : optimization, Distributed Query plan, Honey Bee Mating Optimization HBMO algorithm.

Table des matières

Table des figures

Introduction générale

Depuis plus de quarante années, l'accès aux données demeure toujours un sujet important dans le monde informatique. Beaucoup de travaux ont été réalisés pour faciliter l'accès aux diverses données réparties partout dans le monde, d'une manière efficace et stable.

Les règles d'exécution et les méthodes d'optimisation de requêtes définies pour un contexte centralisé sont toujours valables, mais il faut prendre en compte d'une part la répartition des données sur différents sites et d'autre part le problème du coût des communications entre sites pour transférer les données.[**?**]

La complexité d'une requête dans une base de données répartie est définie en fonction des facteurs suivants :

— Entrées/ Sorties sur les disques (disks I/Os), c'est le coût d'accès aux données.
— Coût CPU : c'est le coût des traitements de données pour exécuter les opérations algébriques (jointures, sélections ...).
— Communication sur le réseau : c'est le temps nécessaire pour échanger un volume de données entre des sites participant à l'exécution d'une requête.

Notons que nous faisons la distinction entre le coût total et le temps de réponse global d'une requête :

— Coût total : c'est la somme de tous les temps nécessaires à la réalisation d'une requête. Dans ce coût, les temps d'exécution sur les différents sites, les accès aux données et les temps de communication entre les différents sites qui entrent en jeu.
— Temps de réponse global : c'est le temps d'exécution d'une requête. Comme certaines opérations peuvent être effectuées en parallèle sur plusieurs sites, le temps de réponse global est généralement inférieur au coût total.

De nos jours, en n'importe quel domaine d'activité ou de recherche, on doit manipuler un grand volume, toujours croissant, de données. En même temps, il est de plus en plus important d'accéder aux informations stockées et de récupérer au plus vite les résultats souhaités.Il est donc nécessaire de mettre en place des mécanismes pour l'optimisation des requêtes distribués.

Parmi l'ensemble des méthodes d'optimisation existantes, les méta - heuristiques résolvent les problèmes de manière générique. Ces algorithmes permettent d'agréger différentes instances de problèmes sans modification fondamentale de leur fonctionnement. Ils sont essentiellement utilisés pour des problèmes d'optimisation, pour lesquels il est actuellement impossible de garantir la meilleure solution en un temps raisonnable. Ils fournissent une solution approchée acceptable en possédant une composante aléatoire. Les méta-heuristiques les plus populaires sont basées sur des analogies avec la biologie (algorithmes évolutionnaires), la physique (recuit simulé) où

l'éthologie (essaim particulaire, colonie d'abeilles artificielles).

La plupart des axes de recherche proposés dans la littérature se concentrent sur la compréhension du comportement des méta-heuristiques. Ils étudient leur fonctionnement intrinsèque, le calibrage convenable de leurs paramètres, leur auto-adaptation.

Ce mémoire s'articule autour du plan suivant :

Le chapitre 1 présente quelques définitions et l'identification des différentes Phases de traitement d'une requête et une introduction au problème d'optimisation des requêtes, déscription des différents modèles d'optimisation des Requêtes, puis s'attarde sur un état de l'art de plusieurs méta-heuristiques à population.

Le deuxième chapitre contient certaines notions et généralités sur le mode de vie des abeilles qui sont indispensables, pour bien comprendre le principe de l'algorithme HBMO et l'algorithme d'optimisation du plan de requête distribué(DQP) avec HBMO.

Le troisième chapitre, présente la Réalisation et aspect d'implémentation Du l'approche HBMO suivie par les résultats expérimentaux.

A la fin de ce manuscrit, nous présenterons une conclusion générale ainsi que les perspectives envisageables.

Chapitre 1

problème d'optimisation des Requêtes distribués

1.1 Introduction

Nous allons présenter, dans ce chapitre quelques définitions et l'identification des Différentes Phases de traitement d'une Requête et une Introduction au problème d'optimisation des requêtes, Déscription des différents modèles d'optimisation des Requêtes, puis s'attarde sur un état de l'art de plusieurs méta-heuristiques à population pour resoudre le problème d'optimisation .

1.2 Définitions

1.2.1 Système réparti

Un système réparti est un ensemble de processeurs autonomes, reliés par un réseau de communication, qui coopèrent pour assurer la gestion des informations.

Le principe est simple : les données et traitements sont répartis sur

différents sites interconnectés par un réseau de communication. Ainsi, la défaillance d'un site ne peut entraîner l'indisponibilité totale du système et sa probabilité peut être négligée grâce à la tolérance aux fautes, assurée par la redondance des informations et des traitements.

L'autonomie des sites est préservée par ce genre de système, en permettant à un groupe d'utilisateurs de créer et de gérer leur propre base de données tout en autorisant les accès aux autres utilisateurs via le réseau.

Un système réparti peut sensiblement améliorer les performances des traitements. En effet, avec une localisation des données et une répartition des traitements bien étudiées, la déperdition induite par les communications des données inter-sites peut être compensée par le gain (*temps de réponse*), issu du parallélisme dans l'exécution des traitements.[**?**]

1.2.2 bases de données réparties

Une base de données répartie BDR est une collection de bases de données localisées sur différents sites, généralement distants, mises en relations les unes avec les autres à travers un réseau d'ordinateurs, perçues pour l'utilisateur comme une base de données unique.

Elle permet de rassembler des données plus ou moins hétérogènes, disséminées dans un réseau sous forme d'une base de données globale, homogène et intégrée.[**?**]

1.2.3 Système de gestion des bases de données réparties

Le SGBDR repose sur un système réparti qui est constitué d'un ensemble de processeurs autonomes appelés sites (*micro-ordinateurs, stations de travail, ... etc.*) reliés par un réseau de communication qui leur permet

d'échanger des données.[**?**]

Un SGBDR suppose en plus que les données soient stockées sur deux sites au moins. Ceux-ci, étant dotés de leur propre SGBD.

Un SGBDR doit offrir une gestion des priorités, des verrous et de la concurrence d'accès de la même façon qu'un SGBD monolithique. Pour cela, il doit disposer de :

— Dictionnaire de données réparties,
— Traitement des requêtes réparties,
— Communication de données inter sites,
— Gestion de la cohérence et de la sécurité.

Le SGBDR assure la décomposition des requêtes distribuées en sous requêtes locales envoyées à chaque site. La décomposition prend en compte la localisation des données pour atteindre une base de données distante.

1.2.3.1 Différentes Phases de traitement d'une Requête

Le traitement d'une requête peut être décomposé en :

1. Analyse :
   — analyse syntaxique et sémantique de la requête,
   — translation de la requête en langage algébrique sous forme d'un arbre : arbre algébrique.
2. Optimisation (phase statique) :
   — générer les différents arbres algébriques équivalents (restructuration),

— traduit les références conceptuels (relation) en références physiques (fichier),

— traduit les opérateurs algébriques en opérateurs de base,

— Evaluer chacun de ces arbres,

— déterminer l'arbre optimale : plan d'exécution

3. Exécution :

   évalue le plan d'exécution.

1.2.4 Introduction au problème d'optimisation des requêtes

La nature déclarative des langages d'interrogation des bases de données, c.-à-d. que ces langages ne fournissent aucune restriction à propos des chemins d'accès aux données, ainsi que l'évolution des structures d'optimisation, conduisent à l'existence d'une multitude d'alternatives, pour exécuter une requête aussi simple soit elle. Ces alternatives sont toutes équivalentes en termes de résultat final, mais peuvent avoir des coûts très différents. Ces coûts se traduisent par les ressources qu'elles utilisent pour exécuter la requête, d'où la nécessité d'optimiser les requêtes.

Cependant, l'augmentation de la complexité des requêtes et du nombre de requêtes à traiter simultanément rend la tâche d'optimisation très coûteuse en temps . D'où le problème d'optimisation des requêtes (*Query Optimization Problem QOP*).[**?**]

1.2.5 Déscription des différents modèles d'optimisation des Requêtes distribués

Il existe essentiellement deux modes d'optimisation, le premier à base de règles et le second à base de coûts. Le premier bien qu'il soit facile à mettre

en oeuvre, reste cependant très simpliste et donc peu fiable. Le second en l'occurrence est beaucoup plus fiable, mais il est lent. Suite à cela un troisième mode a été proposé ces dernières années. Il se base sur l'exploitation des plans d'exécution antérieurs afin de prédire les plans futurs (sur la base d'une fonction de similarité). Quelques SGBD l'ont déjà adopté mais sous des formes simples (à l'exemple d'Oracle qui l'utilise dans un cadre très restreint qui est celui de la similarité syntaxique).[**?**]

1.2.5.1 Mode d'optimisation à base de règles

Dans cette approche (considérée comme la plus ancienne), un ensemble de règles prédéfinies est appliqué par l'optimiseur afin de générer un plan d'exécution, considéré comme intéressant pour une requête donnée. Ces règles, triées par ordre de priorité, indiquent les chemins d'accès et les méthodes de jointure à utiliser selon les structures d'optimisation disponibles.

Bien que ce type d'optimisation soit facile à mettre en oeuvre, il néglige cependant tous les paramètres physiques de la base de données (taille des tables, facteurs de sélectivité des prédicats de sélection et de jointure, etc.).

1.2.5.2 Mode d'optimisation à base de coût

Dans cette technique, un modèle mathématique est utilisé afin d'estimer le coût de chaque plan.

La solution optimale est le plan ayant le coût minimum. Cette optimisation est plus rigoureuse car elle prend en compte les données physiques de la base.

Cette technique est celle utilisée actuellement par tous les SGBD. Ce qui diffère, se sont les mécanismes utilisés pour estimer le coût d'un plan d'exé-

cution. Ces procédés sont maintenus secrets par les concepteurs des SGBD, car se sont eux qui déterminent la qualité de l'optimiseur de requêtes.[**?**]

1.3 Etat de l'art

Il s'agit de présenter ici un ensemble de concepts relatifs à l'optimisation difficile. Plus précisément, cette partie contient un état de l'art autour des problèmes d'optimisation continue, mono-objectif, des termes utilisés et des différentes approches de résolution par méta-heuristique.

Notions abordées dans le chapitre :

— Problèmes d'optimisation statique, mono-objectif, continue ;
— Méta-heuristiques à population ;
— Algorithmes évolutionnaires et de la famille d'intelligence en essaim ;
— Régression linéaire ;
— Méthode des effets élémentaires, méthode de Morris ;

1.3.1 Généralités sur l'optimisation et les méta-heuristiques

Un problème d'optimisation est défini comme la recherche, dans un espace de solutions, d'une solution optimale quantifiée par une fonction objectif. Cette quantification conduit à vouloir maximiser ou minimiser le problème. Un ensemble de méthodes exactes permet de trouver une solution en un temps fini et, de manière générale, polynomial. Ces méthodes, parmi lesquelles on peut citer la méthode du Gradient, la méthode de Newton, la méthode Séparation et évaluation (Branch and Bound) etc., sont déterministes : elles fournissent le meilleur résultat attendu en un nombre

fini d'étapes, par dérivation ou par parcours de tout ou partie des solutions.

L'optimisation difficile représente une classe de problèmes d'optimisation qui ne peut être résolue en un temps polynomial ou par une méthode exacte, sous la contrainte de caractéristiques de la fonction objectif (non convexité, continuité, dérivabilité...). Elle regroupe différentes typologies de problèmes :

- des problèmes à variables continues (non dénombrables), discrètes (dénombrables), ou mixtes ;
- des problèmes uni-modaux ou multimodaux (possédant une ou plusieurs valeurs optimales) ;
- des problèmes mono, multi-objectifs (plusieurs objectifs souvent contradictoires), sous contraintes (faisabilité d'une solution) ;
- des problèmes d'optimisation statique ou dynamique, dans lesquels la fonction objectif, le domaine de définition peuvent restent identiques ou évoluer au cours du temps.

La résolution de cette classe de problèmes se fait alors par des méthodes approchées ou heuristiques, contenant généralement une composante aléatoire. Ces méthodes fournissent rapidement une valeur acceptable, sans garantie d'optimalité. Elles sont communément dédiées à un type de problème.

Lorsqu'une heuristique est généralisable à plusieurs typologies de problèmes sans modification significative, on parle alors de méta-heuristique.

Les méta-heuristiques sont des algorithmes itératifs, possédant une composante aléatoire et parcourant l'espace de recherche par différentes tech-

niques de génération de solutions. Ces algorithmes sont souvent inspirés par des systèmes physiques, biologiques ou éthologiques. Le caractère méta'tient du fait qu'un même algorithme peut agréger différents problèmes d'optimisation difficile sans modification structurelle majeure. La partie dédiée au problème tient essentiellement en la représentation du problème et l'adaptation des opérateurs de recherche.[**?**]

La figure 1.1 présente une fonction de coût à variable continue, définie sur le domaine de recherche. On considère un ensemble de six solutions formant une population. Le problème de minimisation de cette fonction possède une valeur optimale, l'optimum global x^*.

Il possède également plusieurs optima locaux pouvant tromper la recherche.

Les méta-heuristiques vont itérativement déplacer ces solutions selon différentes stratégies afin de trouver une valeur approchée de x^*.

Généralement une méta-heuristique utilise des informations fournies par son voisinage propre (elle va se déplacer dans la direction qui améliore le résultat de la fonction), par un voisinage de solutions proches ou bien par d'autres bonnes solutions déjà trouvées.

Ces algorithmes réalisent le parcours du domaine de recherche selon deux comportements dichotomiques.

Le premier est l'exploration, c'est-à-dire la capacité de l'algorithme à découvrir de nouvelles régions de l'espace de recherche. Ce comportement permet de ne pas être bloqué sur un optimum local mais ne favorise pas la convergence. Le second est l'exploitation : il s'agit de l'aptitude de la

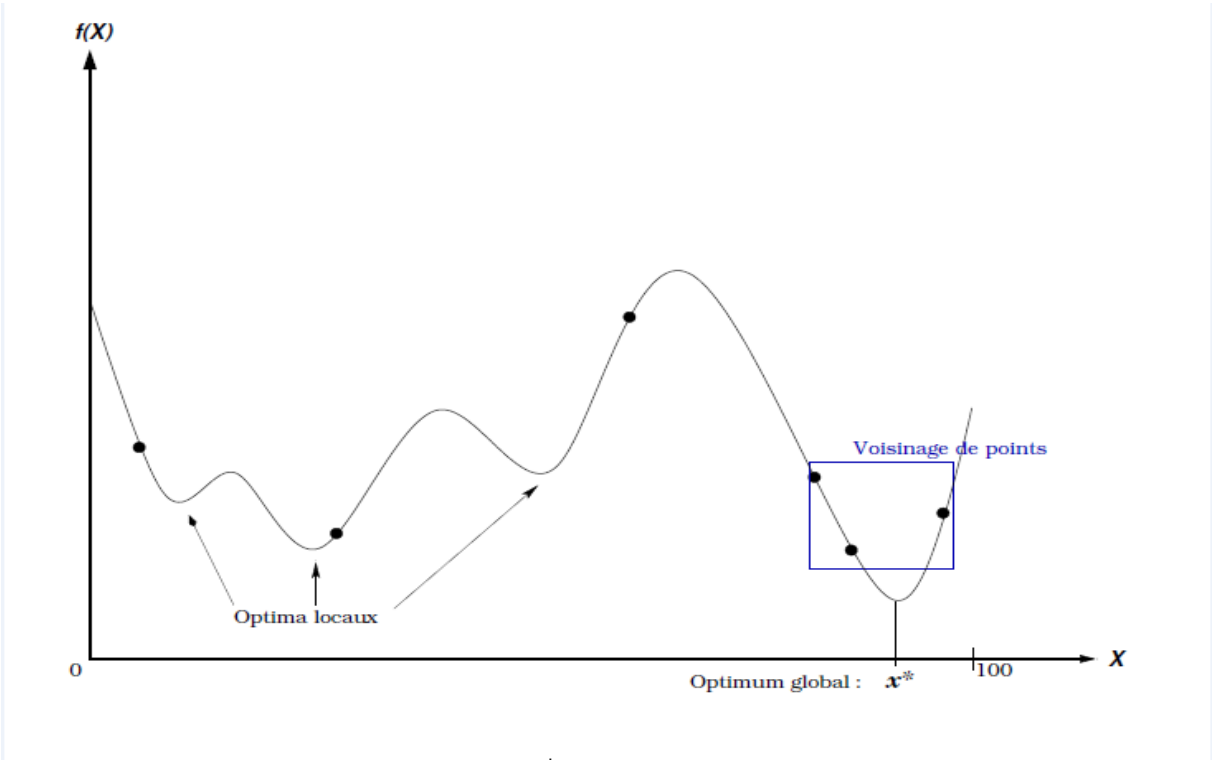

FIGURE 1.1 – *Problème de minimisation par voisinage.*

méta-heuristique, utilisant une bonne solution, à continuer à chercher dans cette zone pour favoriser la convergence.

Le risque est alors de provoquer une convergence prématurée, par exemple en attirant toutes les solutions vers un optimum local. Pour une méta-heuristique, la difficulté est de réaliser un équilibre correct entre ces deux comportements afin de converger vers l'optimum global de l'espace de recherche, tout en évitant de rester bloqué sur une valeur d'optimum local.

Parmi les méta-heuristiques d'optimisation globale les plus populaires, développées pour résoudre des problèmes d'optimisation discrète, on trouve des algorithmes de recherche locale, mono-agent (i.e. se basant sur une unique solution, modifiée itérativement), comme le recuit simulé (Kirkpatrick[Kirkpatrick et collab. 1983])Et la recherche Tabou (Glover [Glover, 1986]). Ces méta-heuristiques ont la capacité de s'extraire d'une solution

minimum locale et ainsi permettre de continuer à explorer le domaine de recherche de la fonction vers une meilleure solution.[**?**]

Nous allons présenter un ensemble de méta-heuristiques à base de population, appartenant à deux familles, les algorithmes évolutionnaires et l'intelligence en essaim, ainsi que leurs améliorations proposées dans plusieurs travaux de recherches.

1.3.2 Algorithmes évolutionnaires

Inspiré de la génétique et de la théorie de l'évolution, l'expression algorithmes évolutionnaires se décline en différentes sous-catégories, agrégeant des problématiques distinctes :

— la programmation évolutionnaire, issue des travaux de Fogel [Fogel et collab, 1966], vise à faire évoluer des structures d'automates finis par croisements et mutations successifs ;
— les algorithmes génétiques, méta-heuristiques pour l'optimisation discrète, issus des travaux de Holland [Holland, 1975], où la solution est souvent représentée en nombre binaire ;
— la programmation génétique, développée par les travaux de Koza [Koza, 1989, 1990], est une méthode de création automatique de programmes, où les chromosomes sont des programmes informatiques ;
— l'évolution différentielle, une méta-heuristique pour l'optimisation continue, de Price et Storm [Storm et Price, 1997] basée sur la mutation, le croisement et la sélection.
— les stratégies évolutionnaires, techniques d'optimisation continue, sont issues des travaux de Rechenberg [Rechenberg, 1989] et Schwefel

[Schwefel, 1981], où une partie des solutions d'une population survit et génère de nouvelles solutions en utilisant le principe de mutation. Ces stratégies sont popularisées notamment par l'algorithme CMA-ES [Hansen et Ostermeier, 2001] de Hansen et Ostermeier. Les termes utilisés en programmation évolutionnaire appartiennent au champ lexical de la génétique.

Une solution à un problème est un individu, une expression du codage de la donnée est un chromosome, une sous-partie de ce codage, un gène. Un ensemble considéré de solutions est appelé une population. Les itérations faisant évoluer la population sont des générations. La population évolue grâce à un ensemble d'opérations : sélection, croisement, mutation. Les individus de la population impliqués dans une opération sont les parents, les individus résultats sont les enfants ou la progéniture.
Dans le cadre de l'optimisation, nous allons présenter le principe des algorithmes génétiques ainsi qu'une étude de l'évolution différentielle.[**?**]

1.3.2.1 Algorithme Génétique

Holland [Holland, 1975] puis Goldberg [Goldberg et Holland, 1988] développent l'algorithme génétique, une méta-heuristique permettant à l'origine de résoudre des problèmes à variables discrètes.C'est un algorithme élitiste à base d'une population de solutions. Cette population évolue durant plusieurs générations en sélectionnant, à chaque étape, les individus les plus performants. Certains individus se reproduisent, d'autres sont supprimés, un héritage génétique est transmis de génération en génération et

conduit les individus les plus adaptés (i.e. répondant le mieux au problème) à survivre.[**?**]

Le processus est répété jusqu'à un certain critère d'arrêt. Il existe de nombreuses adaptations pour le contexte continu, RCGA (Real-Coded Genetic Algorithme) [Chelouah ET Siarry, 2000a ; Davis, 1991 ; Michalewicz, 1996 ; Wright, 1991]. Par ailleurs, les stratégies évolutionnaires et l'évolution différentielle sont deux méthodes fondamentalement dédiées à l'optimisation en variables continues.[**?**]

1.3.2.2 Évolution Différentielle

L'algorithme à évolution différentielle (Differential Evolution, DE) est inspiré des stratégies évolutionnaires et des algorithmes génétiques, et applicable à des problèmes à variables continues. Il a été proposé par Rainer Storn et Kenneth Price en 1997 [Storn et Price, 1997]. Il met en oeuvre trois opérations issues des algorithmes évolutionnaires : la mutation, le croisement et la sélection. La méthode de génération d'un nouvel individu se fait en trois temps. En premier lieu, une mutation engendre un individu à partir d'un ensemble d'autres sélectionnés aléatoirement parmi la population.

De multiples schémas de mutation ont été définis et seront détaillés par la suite. Il y a ensuite une phase de croisement, ou différentes stratégies peuvent être employées pour générer un individu issu du croisement entre le parent et l'individu généré précédemment. La sélection par remplacement est alors effectuée.[**?**]

1.3.3 Algorithme d'optimisation par essaims particulaire

L'optimisation par essaims de particules (en anglais Partical Swarm Optimization PSO) peut être considérée comme une métaphore de calcul et de comportement pour résoudre les problèmes distribués, l'origine de cet algorithme s'inspire du monde du vivant grâce à des observations faites lors des simulations informatiques de la nature fournis par des insectes sociaux comme les fourmis, les termites, les abeilles, les guêpes et aussi par les essaims, les troupeaux en se basant généralement sur la reproduction d'un comportement social. Elle est inventée par Russel Eberhart et James Kennedy [39] en 1995. Ces simulations sont basées sur la capacité des individus d'un groupe en mouvement à conserver une distance optimale entre eux et à suivre un mouvement global par rapport aux mouvements locaux de leur voisinage. Elle s'appuie sur le concept d'auto organisation. Les particules peuvent converger progressivement vers l'optimum global grâce à des règles de déplacement très simples (dans l'espace des solutions).[**?**]

L'intelligence distribuée possède les caractéristiques suivantes :

- **Autonomie** : Les individus contrôlent leur propre comportement.
- **Adaptabilité** : Lorsque les individus peuvent détecter des changements dans l'environnement dynamique ensuite adapter de manière autonome leur propre comportement à ces nouveaux changements.
- **Évolution** : La capacité d'utiliser des groupes composés de quelques-uns et ensuite évolués à des milliers d'individus ayant la même architecture de contrôle.
- **Flexibilité** : La capacité d'enlever ou remplacer ou ajouter dynami-

quement un individu.

— **Robustesse** : Il n'existe pas de coordination centrale, ce qui signifie qu'il n'y pas de point de défaillance unique. Le système d'essaim permet la redondance, ce qui est essentiel pour la robustesse.

— **Le parallélisme massive** : Le fonctionnement du système d'essaim est massivement parallèle et distribué. Les individus accomplies les mêmes tâches au sein du groupe. L'architecture de "Intelligence Distribuée" peut-être considérée comme une architecture SIMD.

— **L'auto-organisation** : L'intelligence exposée n'est pas présente chez les individus, mais émerge plutôt d'une certaine manière sur l'ensemble de l'essaim.

1.3.3.1 Algorithme de Colonies de Fourmis

Le terme colonies de fourmis est un terme générique représentant une classe d'algorithmes, initiée par l'algorithme (*Système de Fourmis*) (*Ant System*) de Colorni, Dorigo et Maniezzo [Colorni et collab, 1992].

Dans L'optimisation par colonie de fourmis, les fourmis artificielles d'une colonie coopèrent pour trouver de bonnes solutions à des problèmes distrubués. La conception des algorithmes ACO est basée sur La coopération d'un ensemble d'agents simples qui se communiquent indirectement par la stigmergie. Les bonnes solutions sont une propriété émergée de l'interaction coopérative des agents. Une fourmi artificielle ACO construit progressivement une solution en ajoutant des composants de solutions définies à une solution partielle en cours de construction.[**?**]

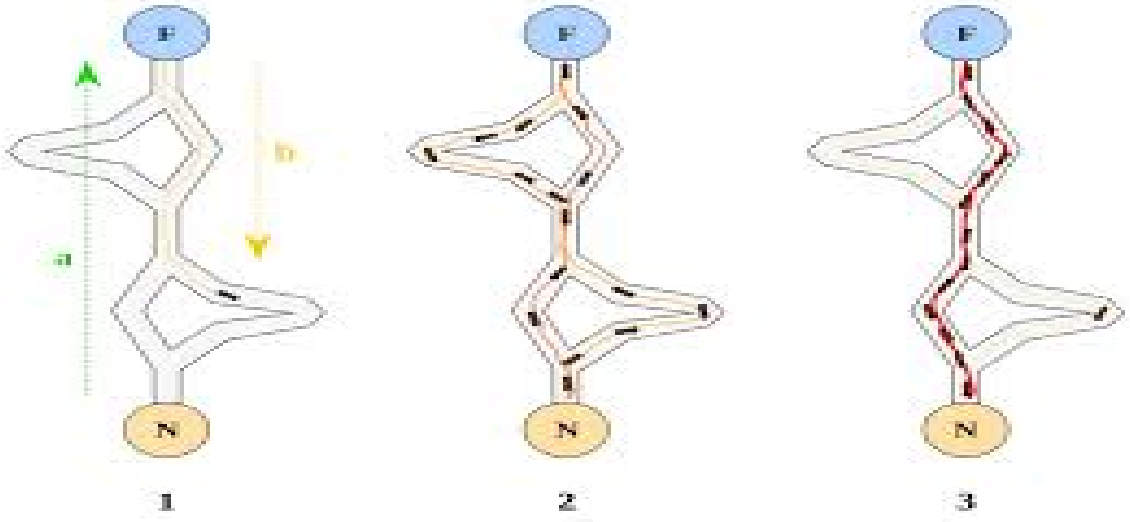

FIGURE 1.2 – *Expérience de sélection du chemin le plus court par une colonie de fourmis (J. Dréo).*

1.3.3.2 Colonie d'Abeilles Artificielles (variante des algorithmes inspirés du comportement des abeilles)

L'algorithme de colonie d'abeilles artificielles ou Artificial Bee Colony (ABC) a été introduit par Dervis Karaboga [Karaboga, 2005] et développé depuis 2005 par Karaboga et Basturk [Karaboga et Basturk, 2008] pour les problèmes d'optimisation continue. C'est un algorithme à population, d'inspiration naturaliste, basé sur le butinage des abeilles.

L'optimisation par colonie d'abeilles est une méta-heuristique s'inspirant du comportement réel des abeilles. Dans la littérature les comportements intelligents des abeilles ont fait l'objet des plusieurs recherches dans le domaine de l'optimisation, en particulier le comportement intelligent de Mating flight des abeilles ou le phénomène de l'évolution des abeilles HBMO et le comportement intelligent de recherche de nourriture forage) BCO.

Dans notre travaille nous nous intéressons à l'adaptation du compor-

tement de l'évolution des abeilles HBMO au problème de traitement et d'optimisation de plan de requête distribuer de temps réel et plus optimal. Ce comportement est formalisé en algorithme connu sous l'acronyme HBMO.[**?**]

Dans ce mémoire on s'intéresse aux l'approche HBMO donc il sera plus détaillé dans ce qui suit.

Chapitre 2

L'approche HBMO

2.1 Introduction

L'optimisation par l'accouplement d'abeilles (HBMO : honey bee mating optimization.) est une nouvelle méta-heuristique inspirée du processus biologique de reproduction des abeilles. Cette méta-heuristique est classée parmi les méthodes évolutives.

Pour bien comprendre le principe de l'algorithme HBMO certaines notions et généralités sur le mode de vie des abeilles sont indispensables de définir pour bien comprendre le processus d'optimisation du plan de requête distribué (DQP) avec l'approche HBMO comme il sera plus détaillé dans ce qui suit.

2.2 Les algorithmes d'accouplement des abeilles

2.2.1 Historique

Au cours de la dernière décennie, les algorithmes d'abeilles, inspirés de la nature, sont Devenus un outil de résolution de problèmes prometteur et puissant. Malheureusement, On ne parvient pas à connaître la date

exacte de la première apparition des algorithmes D'abeilles. Ce qui est sûr, cependant, c'est qu'ils ont été développés en quelques années de façon indépendante par différents groupes de chercheurs. D'après la bibliographie, il semble que l'algorithme HONEY-BEE a été réalisé pour la première fois vers 2004 par CRAIG A.TOVEY à GEORGIA TECH en collaboration avec SUNIL NAKRANI.[**?**]

A la fin de 2004 et au début de 2005, XIN-SHE YANG a développé, à l'Université de CAMBRIDGE, le VIRTUAL BEE ALGORITHM (VBA) pour résoudre des problèmes d'optimisation numérique, cet algorithme permet d'optimiser à la fois les fonctions et les problèmes discrets, cependant ils n'ont donné comme exemples que les fonctions à deux paramètres. Un peu plus tard en 2005, HADDAD , AFSHAR et leurs collègues ont présenté un algorithme du nom de HONEY-BEE MATING OPTIMIZATION (HBMO).

En 2006, B.BASTURK et D.JARABOGO en Turquie, ont développé un algorithme appelé ARTIFICIAL BEE COLONY (ABC) pour l'optimisation de fonction numérique. Nous remarquons ici que la méthode inspirée des abeilles est plus ou moins récente, et qu'avec le temps de nouvelles versions peuvent apparaitre, ce qui rend cette méthode de plus en plus populaire et maîtrisable par les chercheurs.

2.2.2 L'accouplement des abeilles

La nature ne cesse d'inspirer la recherche dans le domaine de l'optimisation. Alors que la génétique, les fourmis et les essaims particulaires en sont des exemples célèbres, d'autres algorithmes d'optimisation inspirés de la nature émergent régulièrement. Dans ce chapitre, nous allons nous

concentrer sur l'un de ces algorithmes, qui est celui des colonies d'abeilles.

L'abeille est l'un des insectes les plus organisés et les plus rigoureux dans son travail. Elle possède une très grande capacité de communication. Et grâce à son intelligence, une méthode appelée méthode des abeilles a été développée. Dans cette méthode, les abeilles artificielles représentent des agents qui, en collaborant les unes avec les autres, résolvent des problèmes complexes d'optimisation combinatoire.[**?**]

2.2.3 Aperçu général sur une colonie d'abeilles

Les abeilles sont des insectes sociaux qui vivent en communauté au sein de la ruche appelée une colonie. En effet, une colonie d'abeilles se compose de faux-bourdons (Drones), de reines (Queens), d'ouvrières (Workers) et des couvains (Broods) comme le montre la figure 2.1. Dans la vie réelle des abeilles, une colonie est généralement constituée d'une seule reine qui survivra dans la ruche, et qui est spécialisée dans la reproduction des abeilles (femelle fertile), zéro à plusieurs milliers de faux-bourdons et habituellement 10.000 à 60.000 ouvrières.

Une reine peut vivre jusqu'à 5 à 6 ans, seule la reine est alimentée par la gelée royale de couleur blanche laiteuse qui fait d'elle la plus grande de toutes les abeilles de la ruche alors que les faux-bourdons et les ouvrières ne vivent jamais plus de 6 mois. Les faux-bourdons représentent les mâles de la colonie d'abeilles, leur fonction est de s'accoupler avec les reines en ajoutant leur sperme à la spermathèque de la reine, mais après le processus d'accouplement ces derniers meurent, tandis que les ouvrières sont considérées comme des abeilles non reproductionnelles (femelles stériles), qui sont spécialisées dans les soins des couvains [Fathian et al. 2007].[**?**]

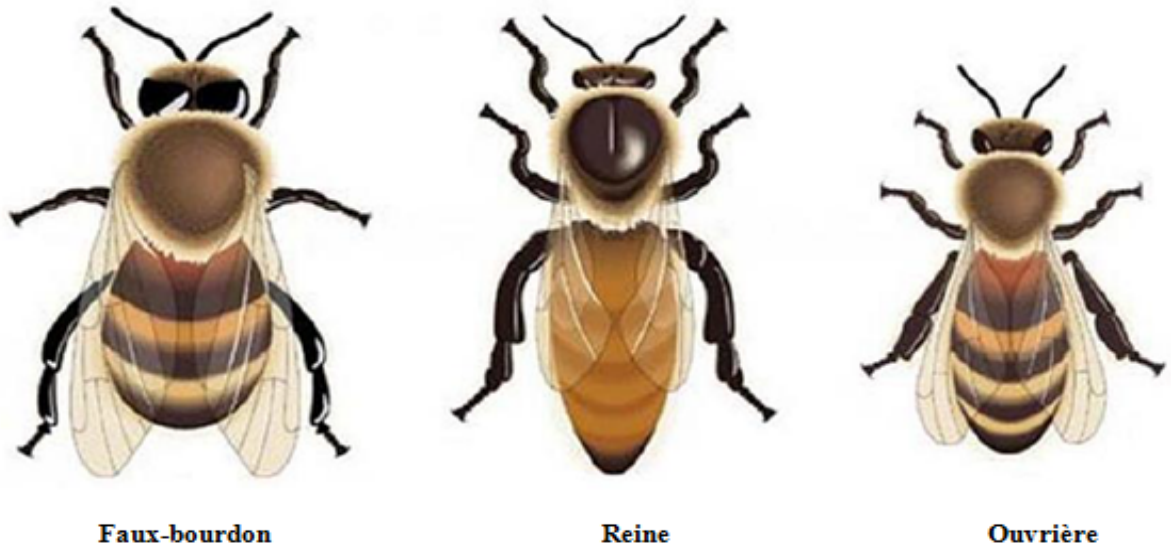

FIGURE 2.1 – *Communautés d'une colonie d'abeilles.*

2.2.4 Principe de fonctionnement de l'algorithme HBMO

L'algorithme HBMO (Honey Bees Mating Optimization) est une méta-heuristique très récente qui fait partie de la classe des algorithmes d'intelligence en essaim dont leur principe est basé sur l'accouplement naturel des reines chez les abeilles, qui a été proposé pour la première fois par Abbass en 2001 [Abbass. 2001]. Lors de la mise en oeuvre de l'algorithme HBMO, l'utilisateur doit définir trois paramètres :[**?**]

— le nombre des reines.
— la taille de la spermathèque qui correspond au nombre maximum d'accouplement par la reine dans un vol nuptial unique.
— le nombre des couvains générés par la reine qui est principalement égal aux nombre correspondant à la taille de la spermathèque de la reine.

Cet algorithme commence par générer aléatoirement une population initiale, sur la base de l'évaluation de la fitness de tous les individus de la population, la meilleure solution est classée comme une reine et le reste représente les faux-bourdons. Le processus d'accouplement des reines se déroule loin de la ruche, où la reine commence son vol nuptial durant lequel les faux-bourdons la poursuivent et accouplent avec elle dans l'air, pendant ce processus la reine s'accouple plusieurs fois (sept à vingt faux-bourdons), ce nombre peut varier selon la saison et la maturité des males disponibles, par contre, les faux-bourdons une seule fois. Après chaque accouplement réussi, les spermatozoïdes du faux-bourdon sont ajoutés à la spermathèque de la reine selon la probabilité **P** donnée par :

$$prob\ (Q, D) = exp^{-\Delta(f)/s(t)}$$

Ou (P) représente la probabilité d'ajouter les spermatozoïdes du faux-bourdon (D) dans la spermathèque de la reine (Q) (probabilité d'accouplement réussie).

Avec Δ(F) est la différence absolue entre la fitness du faux-bourdon (D) et la fitness de la reine (Q)

$$\mathbf{\Delta(F) = |F(D) - F(Q)|}$$

(t) est la vitesse de la reine à l'instant i, La probabilité d'accouplement est élevée lorsque la reine est au début du vol (l'énergie ou vitesse élevée), ou lorsque la fitness du drone est aussi bonne que celle de la reine.

Au début du vol nuptial, la reine est initialisée avec un contenu d'énergie et une vitesse qui sont générées aléatoirement (énergie et vitesse ont le même effet l'une d'entre elles devra être utilisée), et revient à la ruche

lorsque le niveau d'énergie atteint un seuil ou bien lorsque la spermathèque de la reine devient pleine. Le vol nuptial peut être considéré comme un ensemble de transitions dans l'espace de recherche (environnement), où à chaque transition, la reine se déplace avec une certaine vitesse et une certaine énergie qui décroit selon :

$$\mathrm{S(t+1)}=\alpha * S(t)$$
$$\mathrm{E(t+1)}=\alpha * E(t); Ou \alpha \in [0,1]$$

est un facteur de réduction de la vitesse et de l'énergie après chaque transition Une fois que la spermathèque atteint le maximum, la reine retourne à sa ruche et choisit au hasard un faux-bourdon de la spermathèque, ensuite effectue un croisement des génotypes de la reine et du faux-bourdon sélectionné pour générer des couvains, dans l'algorithme HBMO la fonctionnalité des ouvrières est de prendre soin des couvains et de les nourrir par la gelée royale (aliment spécial de la reine) afin de les rendre plus potentiel d ?être la prochaine reine, par conséquent, chaque ouvrière est représentée comme une heuristique qui agit pour améliorer et/ou prendre soin de l'ensemble des couvains en employant une recherche locale, si le couvain généré est plus performant que la reine, il la remplace et un autre vol est lancé avec la nouvelle reine [Afshar et al. 2007, Haddad et al. 2006, Kang et al. 2010]. Le principe de fonctionnement de l'algorithme HBMO se résume principalement en cinq étapes selon la figure 2.1 [Karimi et al. 2014, Horng. 2010] :

— **Vol nuptial des reines avec les faux-bourdons :** l'algorithme

commence avec un vol nuptial, où la reine sélectionne les faux-bourdons (liste des faux-bourdons), selon la règle probabiliste pour former la spermathèque, ensuite un seul faux-bourdon est choisi au hasard pour générer les couvains.

— **Création de nouveaux couvains :** les couvains sont créés par l'opérateur de croisement entre les génotypes de la reine et du faux-bourdon sélectionné.

— **L'utilisation des ouvrières :** les couvains générés sont améliorés par les ouvrières en employant le processus de recherche locale.

— **Adaptation de la fitness des ouvrières :** dans la nature cette étape n'existe pas, l'adaptation de la fitness des ouvrières est basée sur l'amélioration apportée aux couvains, ou la fitness de chaque ouvrière est mise à jour pour donner plus de chance aux ouvrières qui ont un effet plus positif.

— **Remplacement de la reine :** si le nouveau couvain est meilleur que la reine, alors il prend la place de la reine et s'il ne parvient pas à la remplacer, il sera l'un des faux-bourdons dans le prochain vol nuptial.[**?**]

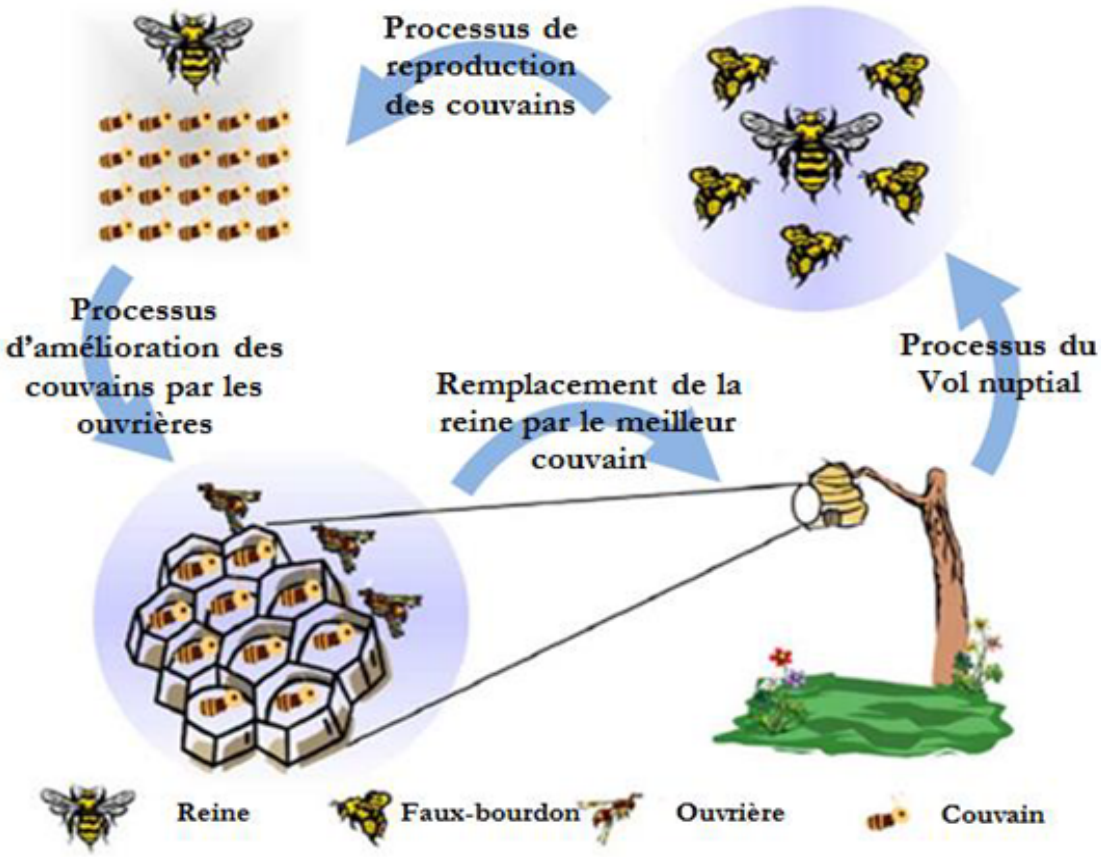

FIGURE 2.2 – *Processus d'accouplement des reines avec les faux-bourdons.*

Cependant l'algorithme HBMO est décrit en détail dans le pseudo-code (Algorithme 1.1) [Sabar et al. 2012, Marinaki et al. 2010] :

Algorithme 1.1 : Algorithme d'accouplement des abeilles (HBMO)

1. Initialisation des paramètres **HBMO**

- $\boldsymbol{\eta}_{sp}$ **:** La taille maximale de la spermathèque de la reine ;
- $\mathbf{S}_q$: La taille de la spermathèque (Spermatheca) de la reine ;
- **QS** $(\boldsymbol{t})$: la vitesse de la reine, **QS** $(\boldsymbol{t}) \in [\mathbf{0.5}, \mathbf{1}]$) ;
- $\boldsymbol{\alpha}$**:** facteur de réduction de la vitesse ($\boldsymbol{\alpha} \in [\mathbf{0}, \mathbf{1}]$) ;
- **F** : Le nombre maximum du vol d'accouplement (*Mating-flight*)
- $\boldsymbol{M}$: Le nombre maximum du vol d'accouplement (*Mating*) ;

2. Génération aléatoire de la population initiale ;
3. Evaluation et classement de la valeur de la fonction fitness de chaque individu de la population(**QPC**) ;
4. Sélection du meilleur individu comme une reine (*min*(**QPC**)) ;

Tant que $\boldsymbol{itr} \leq \boldsymbol{M}$ Faire
  Tant que $\boldsymbol{S}_q < \boldsymbol{\eta}_{sp}$ Faire
    Sélection aléatoire du faux-bourdon (*drone*)

    Si le faux-bourdon satisfait la probabilité d'accouplement
      Faire l'accouplent (croisement entre le faux-bourdon sélectionné et la

2.3 Algorithme d'Optimisation du Plan de requêtes distribuées avec l'HBMO

l'algorithme optimisation du plan de requete distribué avec HBMO est décrit en détail dans le pseudo-code (Algorithme 1.2)[vijay Kumar,Biri Arun, and Lokendra Kumar]

reine → génération des couvains
Ajouter le couvain à liste des couvains
Mise à jour la spermathèque de la reine $S_q = S_q + 1$

Fin Si
Mise à jour de la vitesse de la reine $S(t+1) = \alpha \times S(t)$

Fin Tant que

Amélioration de la fitness des couvains par les ouvrières en utilisant le processus de la recherche locale (iterative improvement) ;
Si la fitness du couvain amélioré est meilleure que celle de la reine

Remplacer la reine par ce couvain
Sinon ajouter ce couvain à la liste des faux-bourdons

Fin Si
Mise à jour de la population des faux-bourdons (remplacer les mauvais faux-bourdons par les meilleurs couvains)
Fin Tant que

5. Retour à la reine (meilleure solution)

Algorithme 2.2: Algorithme d'Optimisation du Plan de requêtes distribuées avec l'**HBMO**

Entrés: la matrice relation-site **R ;**

Nbr de plan de requêtes de drone **D ;**

La vitesse de plan de requête de la reine **QS ;**

La vitesse max de plan de requête de la reine $\mathbf{QS_{max}}$;

La vitesse min de plan de requête de la reine $\mathbf{QS_{min}}$;

Facteur de réduction de la vitesse $\boldsymbol{\alpha}$;

La taille de spermatèque **S ;**

Le nbr des ouvriers **W ;**

Le nombre maximum du vol d'accouplement (*Mating flight*) **F ;**

Sortie : Top-K plan de requêtes ;

Étape 1 : Génération aléatoire de **D** selon **RS** ;
Étape 2 : Evaluation et classement de la valeur de la fonction fitness de chaque

plan de requête d'abeille (**QPC**) ;
Étape 3 : Sélection du meilleur plan de requête comme une reine (*min*(**QPC**)) ;
Étape 4 : Sélection aléatoire de plan de requête de faux-bourdon (*drone*)
Étape 5 : **Répéter**
Répéter// *Phase d'accouplement d'abeille*//

Sélectionnez un plan de requête de drone '**Di**' aléatoirement à partir de '**D**' ;

Calculer **Δ(QPC)=|QPC(Q)-QPC(D)|** ;

Générer un nombre aléatoire **'r'\ r ϵ [0,1]**

$$\mathbf{P(D_i)=\frac{1}{e^{(\Delta(QPC)/QS)}}}$$;

Si $(P(D_i))>'r'$ ajouter **Di** dans **S**

QS=α*QS;

Jusqu'à S est plein ou $\mathbf{QS<QS_{min}}$

Répéter*//Phase de fertilisation //*

Sélectionnez un plan de requête de drone 'Si' aléatoirement de spermathèque S et l'utiliser pour fertiliser le plan de requête Avec le plan de requête de **Q** ;

Jusqu'à l'utilisation de tout les spermathèque de **S**

//Phase de génération de couvain//

Amélioration de la fitness des couvains par les ouvrières en utilisant le processus de la recherche locale (iterative improvement) ;

Si la fitness du couvain amélioré est meilleure que celle de la reine
Remplacer la reine par ce couvain ;
Sinon ajouter ce couvain à la liste des faux-bourdons
Et garder l'ancien plan de requête de la reine ;
Fin Si

Mise à jour des plans des requêtes des faux-bourdons (remplacer les mauvais faux-bourdons par les meilleurs couvains)

Jusqu'à F est terminé

Return Top-K plan de requête des couvains comme **Top-k** plan de requête

Chapitre 3

Réalisation et aspect d'implémentation

3.1 introduction

Dans ce chapitre on implémente notre application en utilisant le langage du programmation **java** avec l'**IDE NetBeans** et le langage de déscription **LATEX**pour le traitement de texte.

3.2 l'envirennement de programmation JAVA

C'est un langage de programmation orienté objet, développé par Sun Microsystems. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitations (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut-être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur.[**?**]

3.3 Présentation l'application

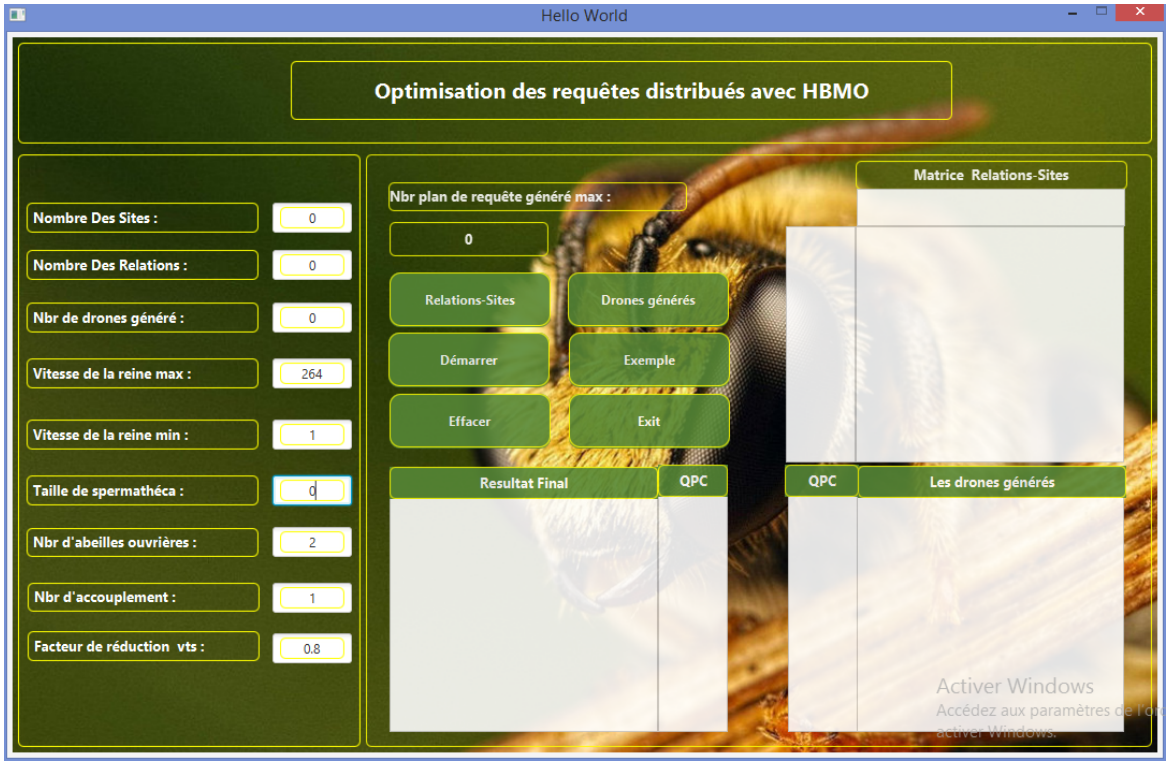

FIGURE 3.1 – *interface principale*

La figure 3.2 représente les paramètres d'HBMO, On a :

— *Le nombre des drones D* : Représente Certains nombre des plans de requête valides pour une requête donnée.

— *Vitesse de la reine max QSmax :* c'est la vitesse de la reine Telque, l'aptitude de chaque plan de requête de drone est calculée et celle avec le moins de la valeur de la fonction fitness de chaque individu de la population(QPC) est choisie pour être le plan de requête de reine de la population initiale.

— *Vitesse de la reine min QSmin :* c'est la vitesse à laquelle la phase d'accouplement s'arrête .

— *La taille de la spermathèque (Spermatheca) S :* c'est le nombre des faux-bourdons d'accouplement réussi.

— *Nombre de vols accouplement :* le faux-bourdon s'accouple une seule fois. Après chaque accouplement réussi, les spermatozoïdes du faux-bourdon sont ajoutés à la spermathèque de la reine selon la probabilité P(Di).

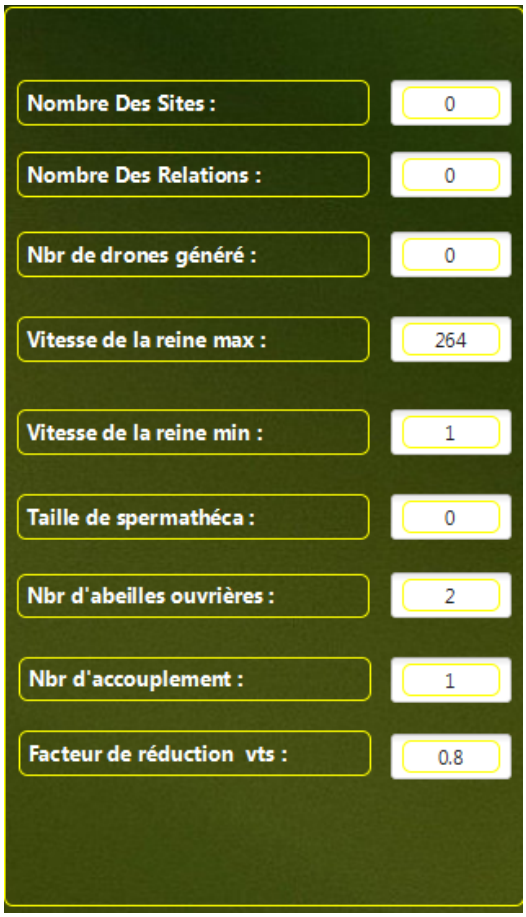

FIGURE 3.2 – *les paramètres d'HBMO*

3.4 Exemple

Considérez la requête distribuée suivante :

Select B1, B2, B3, B4

From R1, R2, R3, R4, R5, R6, R7, R8

Where R1.B1=R2.B1 and R3.B2=R4.B2 and R5.B3=R6.B3 and R7.B4=R8.B4

Le nombre de plans de requêtes distribués possibles augmente de façon exponentielle avec l'augmentation du nombre de relations dans une requête donnée. Il devient impossible pour l'optimiseur de requête de rechercher de manière exhaustive tous les plans de requête distribués. Dans un tel espace de recherche expansif, il est impossible de rechercher le plan de

requête distribué le plus "proche" ayant le coût le plus bas. Ceci, étant un problème d'optimisation combinatoire, peut être résolu en générant des plans de requête qui entraînent moins de coût de traitement des requêtes. Puisque le coût dominant du traitement des requêtes distribuées est le coût dû à la communication inter-site, le but serait de générer des plans de requête impliquant un moindre coût de communication de site à site. Ce coût, modélisé comme le coût de proximité de la requête (QPC) et il est défini comme indiqué ci-dessous :

$$\text{QPC}=\sum_{i=1}^{M} \frac{NS_i}{N}\left(1-\frac{NS_i}{N}\right)$$

où M est le nombre de sites accédés par le plan de requête, NSi est le nombre de fois que le site i est utilisé dans le plan de requête,et N est le nombre de relations.Le coût de proximité de requête (QPC) est utilisé pour mesurer la fitness de ces plans de requête. Les plans de requête ayant des QPC inférieurs sont plus souhaitables que ceux ayant des QPC plus élevés.tel que la fitness de chaque plan de requête de drone est calculée et celle avec le moins de QPC est choisie pour être le plan de requête de reine. Le plan de requête de reine entreprend un certain nombre de vols d'accouplement. la matrice relation-site représente la correspondance entre les relations et les sites. chaque matrice relation-site a un nombre possible de drones de plan de requêtes généré (D) donnée par :

$$D = NS_1 \times NS_2 \times NS_3 \times ... \times NS_n$$

n est le nombre de correspondance de relation-site de chaque relation. Dans cet exemple on a 8 relations et 8 sites avec 21600 drones de plan de requête et 20 plans de requêtes de drones (**figure 3.4**).

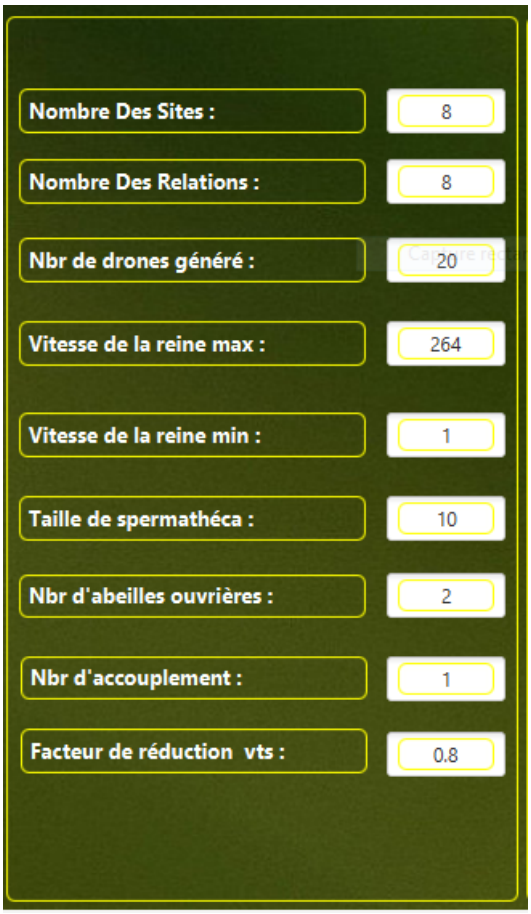

FIGURE 3.3 – *initialisation des paramètres d'HBMO*

3.5 resultats experimentaux

la **figure 3.4** représent la matrice relation-site de la requête distribuée ci-dessus. le resultat d'application d'HBMO est donnée dans la **figure 3.5** et Le QPC des 20 plans de requête générés et les Top-5 plans de requête distribués comme le montre la **figure 3.6**. et la **figure 3.2** représente resultat du Comparison entre les phases de l'algorithme HBMO

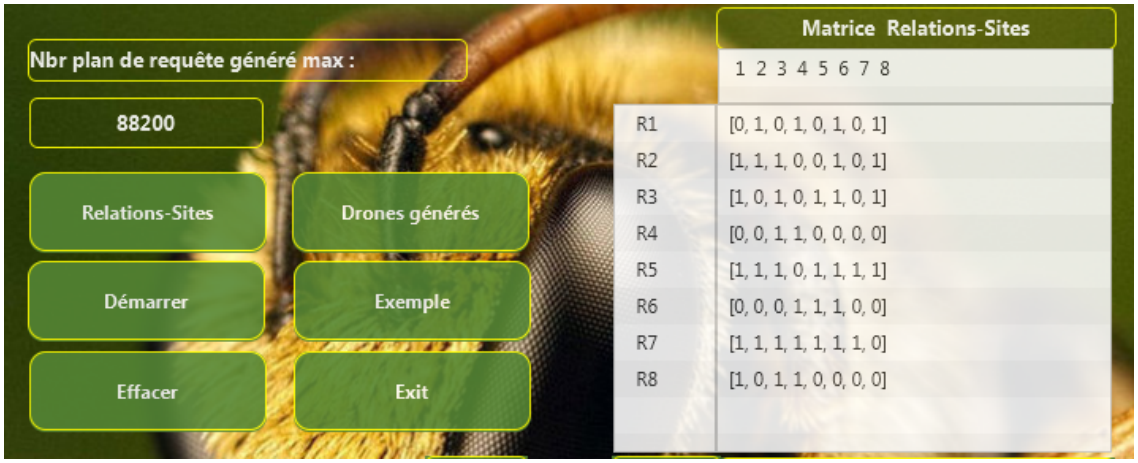

FIGURE 3.4 – *matrice relation-site*

FIGURE 3.5 – *resultat des phases de l'algorithme HBMO*

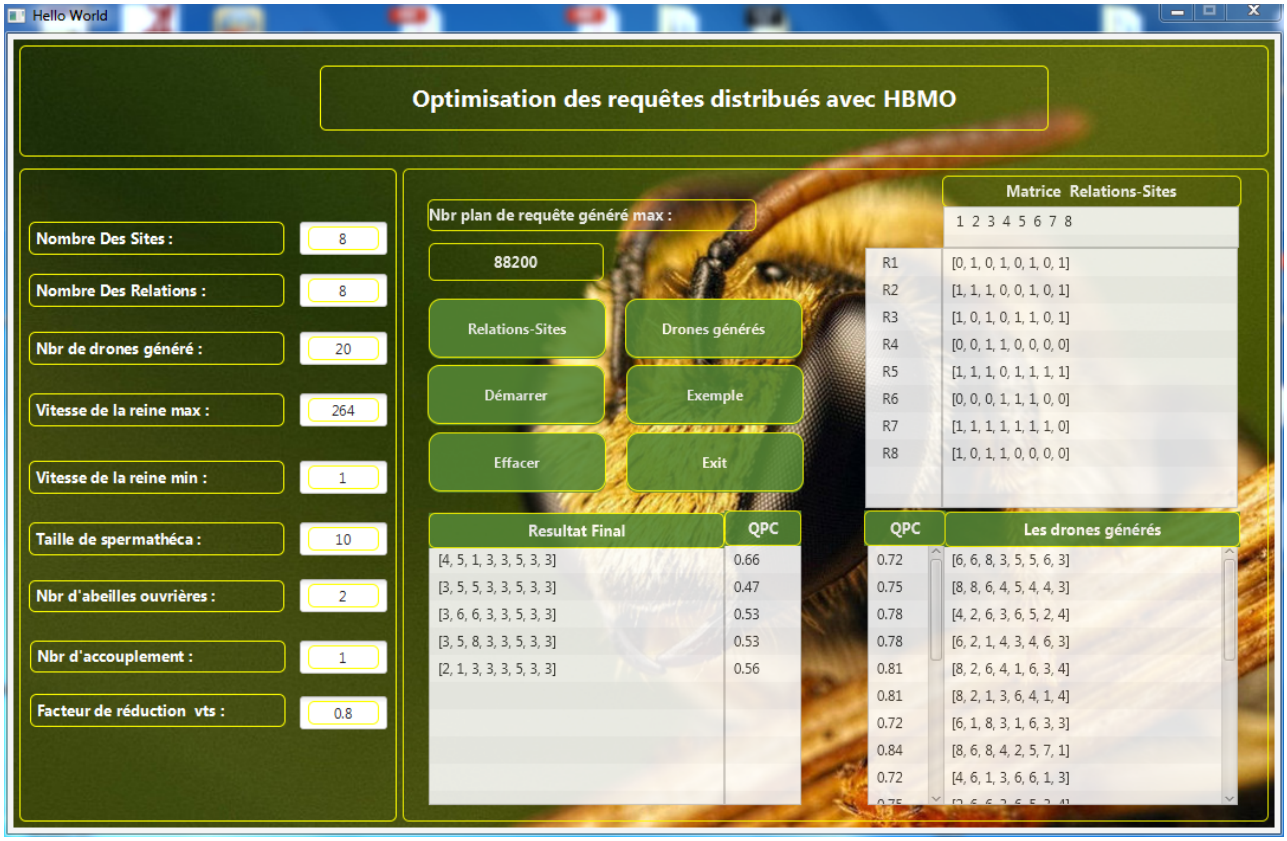

FIGURE 3.6 – *résultat final d'application d'HBMO*

FIGURE 3.7 – *resultat du Comparison entre les phases de l'algorithme HBMO*

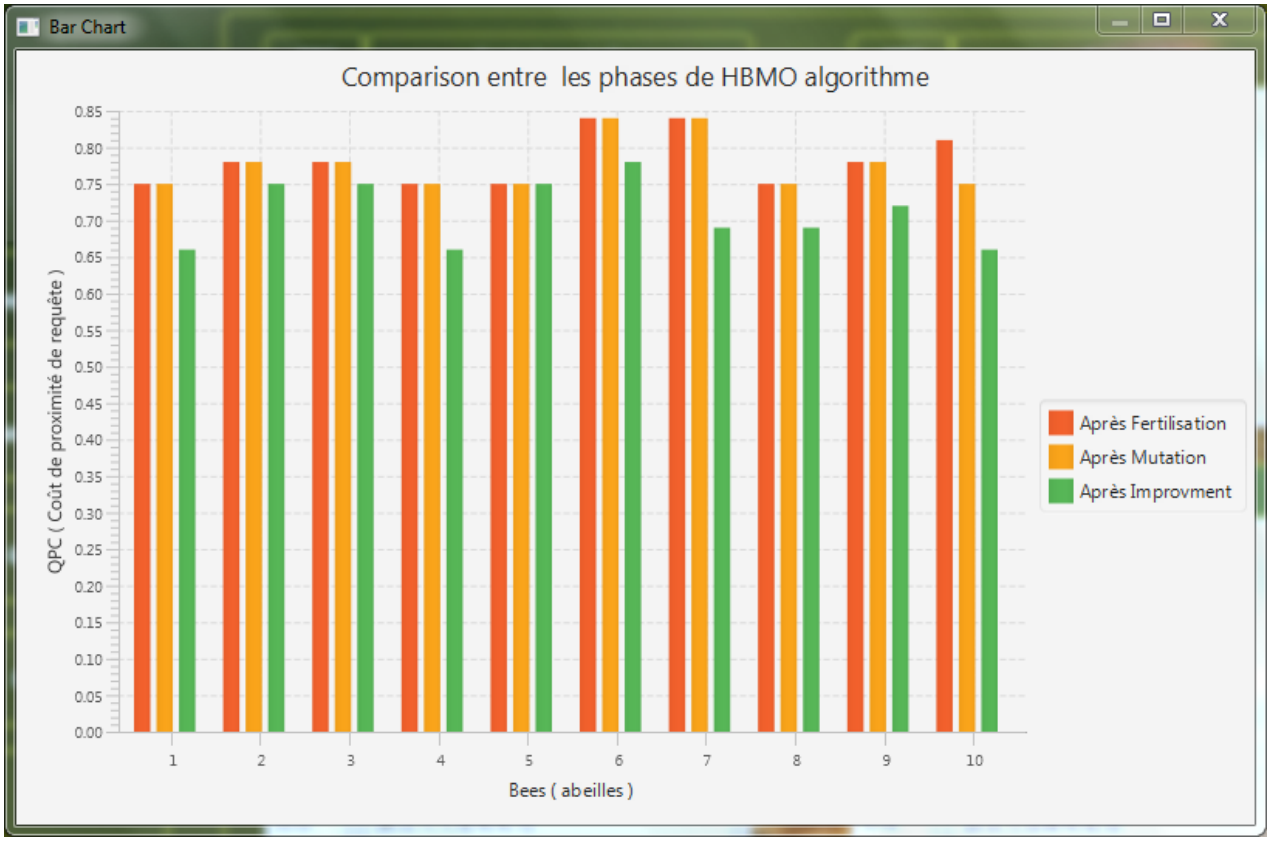

FIGURE 3.8 – *résultat du Comparison entre les phases de l'algorithme HBMO*

Conclusion Générale

Les systèmes distribués gérent le stockage des grandes masses de données, ces systèmes doivent répondre rapidement à une quantité toujours plus importante de requêtes émises par des clients distants.
Donc il faut prendre en compte l'accès aux diverses données réparties sur différents sites et le problème du coût des communications entre sites pour transférer les données.
l'objectif principal consiste à minimiser le temps de traitement des requêtes.
Il est donc nécessaire de mettre en place des mécanismes pour le traitement et l'optimisation des requêtes, parmi l'ensemble des méthodes d'optimisation existantes dans la litterature,on a les méta-heuristiques résolvent les problèmes de manière générique.
Dans ce travail qui nous nous intéressons à l'adaptation et l'implémentation d'une approche méta-heuristiquen insppirée de la nature pour le comportement de l'evolution des abeilles ,HBMO a fin de rémedier le probléme de traitement des requêtes distribués et dans l'objectif d'optimiser la génération des plans des requêtes, en se basant essentiellement sur le critére de coût d'approximité de requêtes (QPC) comme fonction de fitness.
l'implémentation a été réalisé sous l'environnement de programmation **JAVA**, l'exprimentation a montre des resultats satisfaisants.

comme perspective à ce travail, nous envisageons l'ajout d'une procedure d'extraction de la matrice relation-site à partir de la requête utilisateur, ainsi que l'hybridation de notre approche avec d'autre technique d'optimisation comme PSO,AG,DE,...

Bibliographie

[2] https ://www.memoireonline.com/02/11/4278/m$_C$*onception* − *et* − *realisation* − *dune* − *base* − *de* − *donnees* − *repartie* − *sous* − *oracle* − −*cas* − *de* − *lhebergement* − *d*1.*html*.

[1] Noel Gillet, Optimisation de requêtes sur des données massives dans un un environnement distribué .

[3] Peio Loubière, Amélioration des méta-heuristiques d'optimisation à l'aide de l'analyse de sensibilité. le 21 novembre 2016

[4] Ladjel BELLATRECHE et Lamia SADEG, Une approche pour l'optimisation des requêtes dirigée par la réutilisation des plans d'exécution

[5] CHIALI Imane, Détection multi-utilisateurs MUD dans un système multi-antennes et à modulation multi-porteuses MIMO-OFDM par des approches méta-heuristiques. le 05 Octobre 2017.

[6] J. KAABI, C. VARNIER, N. ZERHOUNI. Genetic algorithm for scheduling production and maintenance in a Flow Shop. Laboratory of automatic of Besançon, France. 2003

[7] C.H. PAPADIMITRIOU K. STEIGLITZ, Combinatorial optimization ? algorithms and complexity. Prentice Hall, 1982.

[8] Haddad, O. B., Afshar, A., et Mariño, M. A. Honey-Bees Mating Optimization (HBMO) Algorithm : A New Heuristic Approach for Water Resources

Optimization. Water Resources Management, tome 20, no 5, pages 661 ? 680, 2006. ISSN 0920-4741, 1573-1650.

[9] Wright, A. H. 1991, genetic algorithms for real parameter optimization, dans Foundations of Genetic Algorithms, Morgan Kaufmann, p. 205 ?218.

[10] Wang, H., Z. Wu, X. Zhou et S. Rahnamayan. 2013, ?Accelerating arti ?cial bee colony algorithm by using an external archive ?, dans Proceedings of the 2003 IEEE Congress on Evolutionary Computation, Dec. 8 ?12 2003, Canberra (Australia), IEEE, p. 517 ?521.