MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS DE MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique
Département de Mathématiques et d'Informatique
Filière Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : Ingénierie des Systèmes d'Information

Titre
Optimisation de requete distribuée

Etudiants :

ADDA BENATTIA Ismail
GHELLAL Hamza

Encadrant(e) :

Mme. BETOUATI Fatiha

Année Universitaire 2014/2015

Résumé

L'Optimisation de requête L'optimisation de requêtes distribuée est l'un des problèmes les plus durs dans le secteur de base de données. Le grand succès commercial des systèmes de base de données est partiellement dû au développement de la technologie sophistiquée d'optimisation de requêtes où les utilisateurs posent des requêtes d'une manière déclarative en utilisant le SQL ou OQL et l'optimiseur du système de base de données est chargé de trouver une bonne manière (c.-à-d. plan) à exécuter ces requêtes. Le traitement de requête distribuée implique la transmission de données entre les différents sites via un réseau informatique. Le but du système de base de données répartie est de fournir une stratégie de traitement efficace pour une requête donnée. Dans l'optimisation de requête, un coût est associé à chaque plan d'exécution d'une requête. Ce coût est la somme du coût local (coût E/S, coût CPU à chaque site) et du coût de transfert des données entre les sites par exemple. L'optimiseur est chargé de trouver une bonne manière (c.-à-d. plan) d'exécuter cette requête, par exemple, il détermine quels index devraient être employés et dans quels ordres les opérations d'une requête (par exemple jointure, sélection, projection) devraient être exécutées. À cet effet, l'optimiseur énumère des plans alternatifs, chaque un avec un coût à l'aide d'un modèle de coût.

Dans ce projet, nous étudions le problème de l'optimisation de requête distribuée et les caractéristiques de l'optimiseur nous présentons l'espace de recherche, les stratégies de recherche et nous citerons les travaux que effectués dans ce domaine.

Mots clés : optimisation de requête distribuée, stratégie de recherche, base de données distribuée, stratégie déterministe, stratégie randomisée, modèle de coût.

Remerciements

Nous tenons à remercier en premier lieu, DIEU, le tout puissant, qui nous a donné le courage, la force et la volonté pour bien mener ce modeste travail à terme.

Nous remercions particulièrement notre encadreur Mme BETOUATI Fatiha, pour la confiance qu'il nous a accordée en nous proposant ce sujet, pour ses précieux conseils, son entière disponibilité , ainsi que ses encouragements sans lesquels ce travail n'aurait jamais vu le jour.

Nos remerciements vont également à l'ensemble de l'administration et en particulier nos enseignants du département mathématiques et d'informatique qui nous ont formés durant ces cinq dernières années.

Nous présentons nos gratitudes au président et aux membres du jury qui nous font l'honneur de juger notre travail.

Finalement, nos remerciements vont à nos familles, à tous nos collègues de 2éme année Master en informatique, à tous nos amis et toute personne ayant contribuée de près ou de loin à l'aboutissement de ce modeste travail.

Dédicace

« Louange à Dieu le tout puissant qui ma a aidé à achever ce travail de recherche, et que le salut soit sur son prophète Mohamed ».

Je dédie ce modeste travail, à mes parents qui ont toujours été présent pour moi et qui m'ont toujours soutenu dans ma vie sans vous je ne serai pas là aujourd'hui. Un grand merci,

A mon frère, mes chères sœurs et mes familles.

A ces êtres rares qui m'aident à découvrir chaque jour un peu plus, le bonheur du mot « amitié »...

Ismail

Dédicace

« Louange à Dieu le tout puissant qui ma a aidé à achever ce travail de recherche, et que le salut soit sur son prophète Mohamed ».

Je dédie ce modeste travail, à mes parents qui ont toujours été présent pour moi et qui m'ont toujours soutenu dans ma vie sans vous je ne serai pas là aujourd'hui. Un grand merci,

A mon frère, mes chères sœurs et mes familles.

A mes frères de G.c.ain-Tedeles HAND BALL

et

Tous les gens qui ma aider pour je suis là.

A ces êtres rares qui m'aident à découvrir chaque jour un peu plus, le bonheur du mot « amitié »…

Hamza

TABLE DES MATIERES

CHAPITRE III : L'APPROCHE PROPOSE

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE D'ACRONYMES

BD : Bases de Données.

BDR : Base de Données Répartie.

SGBD : Systèmes de Gestion de Bases de Données.

SQL : Structured Query Language.

PEL : Plan d'exécution logique.

QEP : Query execution plan.

TT : Temps total.

TR : Temps de réponse.

RT : Réponse time.

DBMS : Database Management System.

WAN : Wide Area Network.

SF : Sélectivité facteur.

GA : Génétique algorithme.

TSP : Traveling Salman Problème.

CAO : conception assistée par ordinateur.

QDA : Query distributor Agent.

LOA : Local optimizer Agents.

GOA : Global Optimizer Agent.

E/S : Entrée/sortie.

CPU : Central Processing Unit.

PVC : Problème de Voyageur de Commerce.

IDE : Integrated Development Environment.

CDDL : Common Development and Distribution License.

XML : Extensible Markup Language.

HTML : Hypertext Markup Language.

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Les bases de données réparties peuvent être défini comme une collection de base de données multiple et logiquement en corrélation répartie sur un réseau informatique. Ces bases de données logiquement en corrélation sont gérées par le système de gestion de bases de données réparti (DDBMS) **[1], [10]**. Dans ce système, le traitement d'une requête implique la transmission de données entre les différents sites puisque les données sont géographiquement distribuées.

Généralement l'objectif souhaité dans ce contexte est de réduire le coût de communication exigé pour la transmission de données **[1]**, qui est la clé des performances de ces systèmes de base de données réparties **[11], [12]**. Une requête dans ce système peut être exécutée avec différents plans.

Un optimiseur a fondamentalement trois composants, à savoir, l'espace de recherche, la stratégie de recherche et le modèle de coût **[2]**. Une requête donnée est représentée comme un arbre de requête (arbre de jointure) dans un espace de recherche en utilisant les règles de transformation, si cette requête comporte plusieurs opérateurs et plusieurs relations, alors l'espace de recherche deviendra très grand parce qu'il contiendra un grand nombre d'arbres équivalents .

La stratégie de recherche explore l'espace de recherche pour trouver le meilleur plan d'exécution. Le modèle de coût est employé pour prévoir le coût de chaque plan. Ce modèle se compose de différents types d'éléments, comme le coût de stockage secondaire, le coût de stockage de mémoire, le coût de calcul et le coût de communication **[1] [2] [3] [10].**

Dans le premier chapitre, nous rappelons une généralité sur les bases de données distribuée qui contient les différent architecteur, les types et Les règles de fragmentation…etc.

Dans le deuxième chapitre, nous présentons une étude sur l'optimisation des requêtes et les travaux qui déjà fait pour prend une aidée pour notre projet.

Ensuite dans le troisième chapitre, nous présentons une étude détaillée sur le principe de l'algorithme génétique et le fonctionnement de cet algorithme.

Finalement le quatrième chapitre, nous avons présenté une implémentation détaillée sur le fonctionnement de notre projet.

Chapitre I

Généralités sur LES BASE DE DONNÉES RÉPARTI

Introduction

Une base de données répartie (BDR) est un ensemble de bases de données (BD) manipulés comme s'il s'agissait d'une BD unique. Cependant, chaque BD est gérée par des systèmes de gestion de BD (SGBD) autonomes qui sont localisés sur des sites éventuellement distants **[14]**. Il montre comment est assurée la transparence de la localisation au travers des mécanismes de fragmentation qui explicitent la répartition des données. Il décrit ensuite le processus de décomposition des requêtes en sous-requêtes adressées aux différents sites qui composent l'architecture répartie.

I.1 Généralité sur les bases de données réparties

Une base de données distribuée est gérée par plusieurs processeurs, sites ou SGBD, tout en cachant la complexité des opérations aux utilisateurs qui à leurs tours pensent qu'ils accèdent à une base de données centralisée. Les bases de données distribuées peuvent être stockées dans plusieurs ordinateurs qui sont situés dans un même lieu physique, ou dispersés sur un réseau d'ordinateurs interconnectés (comme dans le Cloud). D'habitude, ces ordinateurs s'appellent « nœuds ». **La Figure 1** illustre ce concept de façon simplifié. Dans les systèmes distribués, il faut faire la distinction entre une base de données distribuée homogène et une autre qui est hétérogène. La première est caractérisée par le fait que toutes les instances ou nœuds démarrent sur un seul logiciel et partagent un seul matériel informatique (hardware), de plus, le système peut être affiché dans une interface unique comme s'il s'agissait d'une seule base de données. Le deuxième type peut avoir différents matériels informatiques (hardware), systèmes d'exploitation, SGBD, et différents schémas de base de données (la structure décrite dans un langage formel qui soutient le modèle géré par le SGBD) **[18]**.

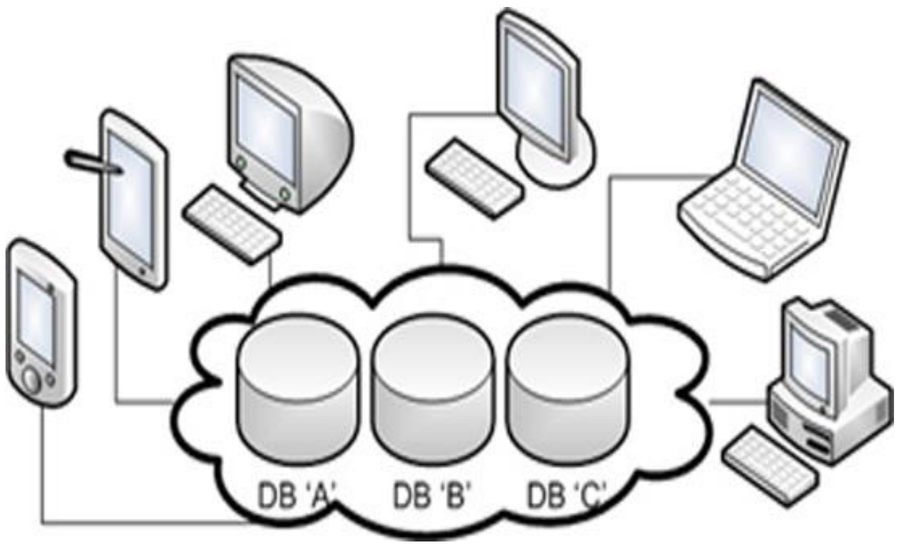


*Figure I.1.Base de données distribuée **[18]**.*

Les bases de données distribuées, également appelées «bases de données réparties », peuvent être confondues avec d'autres types de bases de données comme les bases de données fédérées ou les bases de données parallèles.

I.2 Les différentes architectures

Dans un environnement de base de données répartie, il existe 2 principaux types d'architectures :

I.2.1 L'architecture client-serveur : Dans cette architecture, l'application client se connecte au serveur de base de données (ici Oracle). Ce dernier à son tour, leur renvoie des réponses en fonction de leurs requêtes **[20]**.

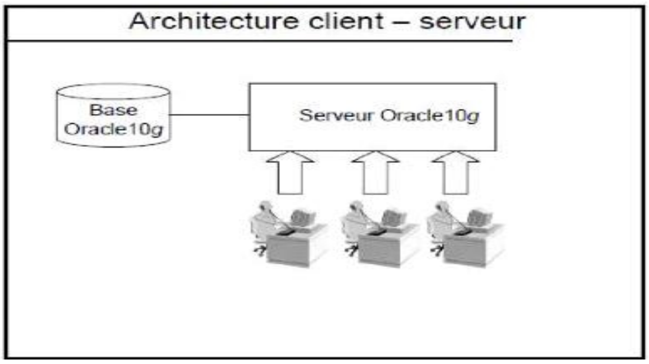


*Figure I.2. Architecture Client-serveur **[20]**.*

I.2.2 L'architecture serveur-serveur : Dans un système de base de données répartie, il existe en général plusieurs serveurs de données qui fonctionnent selon l'architecture suivante : Chaque serveur gère sa base de données et échange les informations avec les autres **[20]**.

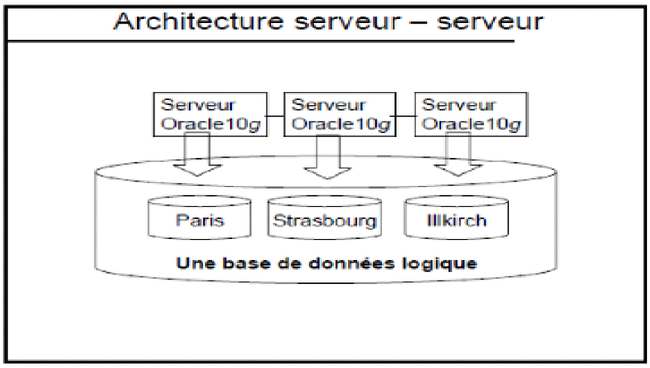


*Figure I.3.Architecture serveur-serveur **[20]**.*

Le tout est vu comme une seule base de données logique. De façon globale, voici comment fonctionne un système de base de données réparties : Les clients se connectent à leurs serveurs respectifs, et ces derniers s'échangent les informations si nécessaires **[20]**.

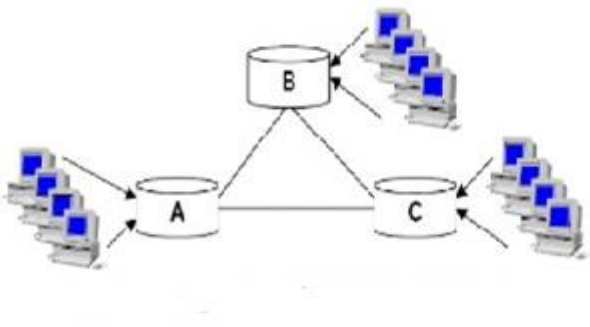


Figure I.4.Architecture général ***[20]****.*

I.3 Conception d'une base de données reparties

Comme dans tous les mécanismes, la phase de conception est la plus importante et elle est déterminante dans la mise en place d'une base de données reparties. Le rôle du concepteur est de définir les différents fragments de la base et leurs localisations ; d'évaluer les différents coûts de stockage et de transfert et les priorités à respecter. On distingue deux principaux types de conception :

I.3.1 La conception ascendante : Suivant la figure, il existe plusieurs bases de données disjointes qu'il faut réunir en une seule base de donnée repartie et cohérente avec un schéma de conception global. Cette approche se base sur le fait que la répartition est déjà faite, mais il faut réussir à intégrer les différentes BDs existantes en une seule BD globale. En d'autre terme, les schémas conceptuels locaux existent et il faut réussir à les unifier dans un schéma conceptuel global **[20]**.

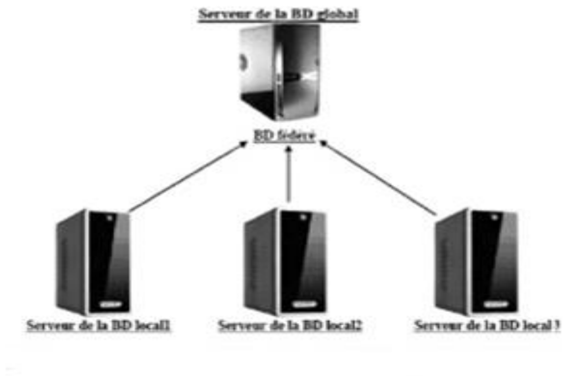


Figure I.5. Architecture de la conception ascendante ***[20]****.*

I.3.2 La conception descendante : Ici, on a au départ une seule base de données qu'il faut fragmenter et allouer les fragments aux différents sites. On va d'un schéma global de conception à des sous schémas locaux.
On commence par définir un schéma conceptuel global de la base de données répartie, puis on le distribue sur les différents sites en des schémas conceptuels locaux. La répartition se fait donc en deux étapes, la première étape est la fragmentation et la deuxième étape est l'allocation de ces fragments aux sites. L'approche top down est intéressante quand on part du néant. Si les BDs existent déjà, la méthode Botton up est utilisée.

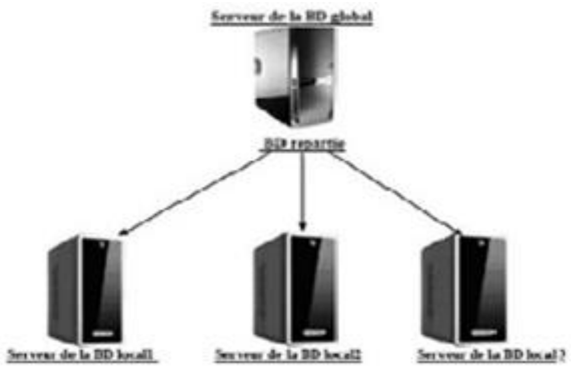


Figure I.6.Architecture de la conception descendante ***[20]***.

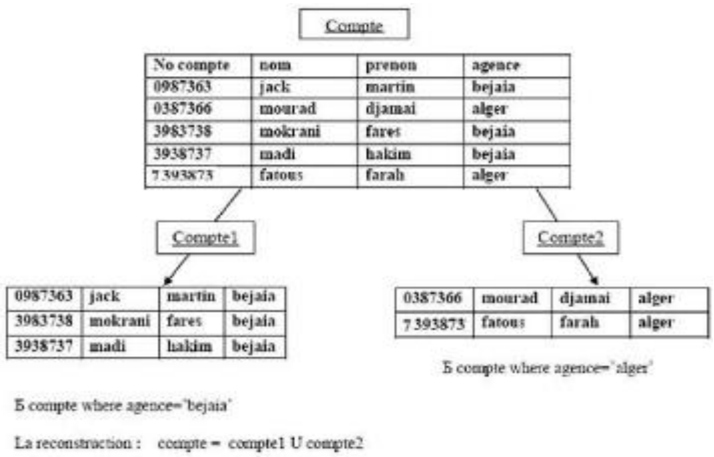
I.4 La fragmentation

I.4.1 Définition : La fragmentation est le processus de décomposition d'une base de données en un ensemble de sous bases de données. Cette décomposition doit être sans perte d'information. La fragmentation peut être coûteuse s'il existe des applications qui possèdent des besoins opposés **[20]**.

I.4.2 Types de fragmentation

a) La fragmentation horizontale

C'est un découpage d'une table en sous tables par l'utilisation de prédicats permettant de sélectionner les lignes appartenant à chaque fragment.
L'opération de fragmentation est obtenue grâce à la sélection des tuples d'une table selon un ou plusieurs critères bien précis, et la reconstitution de la relation initiale se fait grâce à l'union (U) des sous-relations **[20]**.
L'exemple suivant :



No compte	nom	prenom	agence
0987363	jack	martin	bejaia
0387366	mourad	djamai	alger
3983738	mokrani	fares	bejaia
3938737	madi	hakim	bejaia
7393873	fatous	farah	alger

0987363	jack	martin	bejaia
3983738	mokrani	fares	bejaia
3938737	madi	hakim	bejaia

0387366	mourad	djamai	alger
7393873	fatous	farah	alger

Figure I.7.Exemple de fragmentation horizontale ***[20].***

b) La fragmentation verticale

C'est le découpage d'une table en sous tables par projection permettant de sélectionner les colonnes composant chaque fragment. La relation initiale doit pouvoir être recomposée par la jointure des fragments **[20]**.

Exemple :

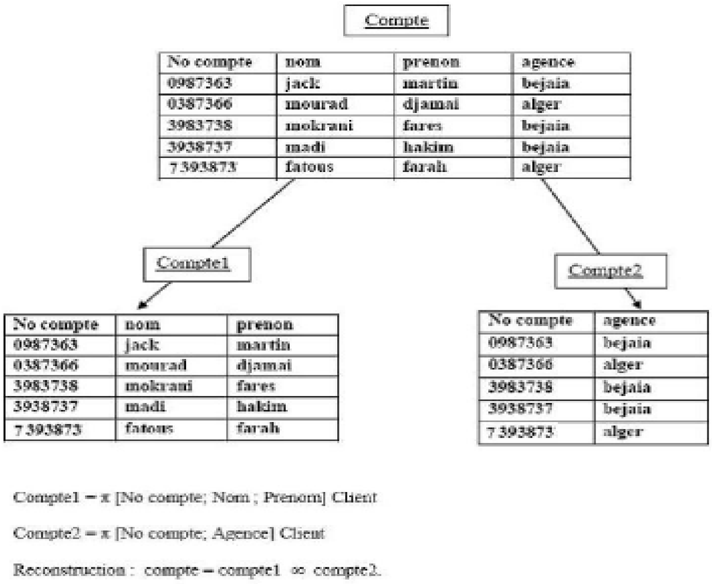


Figure I.8.Exemple de fragmentation verticale ***[20]***.

c) La fragmentation mixte

Elle résulte de l'application successive d'opérations de fragmentation horizontale et verticale sur une relation globale **[20]**.

1.4.3 Les règles de la fragmentation :

Un problème qui se pose pour la fragmentation est comment définir un bon degré de fragmentation. Il existe trois règles pour la fragmentation :

1. ***Complétude :*** pour toute donnée d'une relation globale R, il existe au moins un fragment Ri de la relation R qui possède cette donnée.
2. ***Reconstruction :*** pour toute relation R décomposée en un ensemble de fragments Ri, il existe une opération de reconstruction à définir en fonction de la fragmentation. Pour les fragmentations horizontales, l'opération de reconstruction est une union. Pour les fragmentations verticales c'est la jointure.
3. ***Disjonction :*** une donnée n'est présente que dans un seul fragment, sauf dans le cas de la fragmentation verticale pour la clé primaire qui doit être présente dans l'ensemble des fragments issus d'une relation.

I.5 L'allocation

Lorsque le concepteur a fini de fragmenter sa base, il faut ensuite allouer chaque fragment sur son site correspondant. Cette phase est appelée Allocation. L'allocation peut être faite de plusieurs façons **[20]** :

- **La réplication totale des données** : Pour des raisons de fiabilité, on peut décider de répliquer toutes les données sur tous les sites. Ainsi, si un site est temporairement ou définitivement défaillant, on utilise simplement un autre. Mais cette méthode n'est pas très efficace lorsque les données sont régulièrement mises à jour car le problème de cohérence de données se pose.
- **L'absence de réplication :** On peut aussi choisir de ne rien répliquer afin d'assurer une meilleure cohérence de données. Ici, chaque donnée est mise à jour sur un seul site. Cette méthode est plus efficace quand les données sont beaucoup plus modifiées que lues.
- **La méthode hybride :** Afin de bénéficier des deux méthodes citées à la fois, la méthode hybride peut être utilisée. Ainsi, les données en lecture seulement peuvent être répliquées et non pas les données en lecteur écriture.

I.6 Traitement de requête distribuée

L'objectif du traitement d'une requête distribuée est de transformer une requête de haut niveau sur une base de données répartie en une stratégie d'exécution efficace exprimée en langage de bas niveau sur les bases de données locales. Typiquement, le traitement de requête implique plusieurs étapes comme illustré dans la figure au-dessous **[19] :**

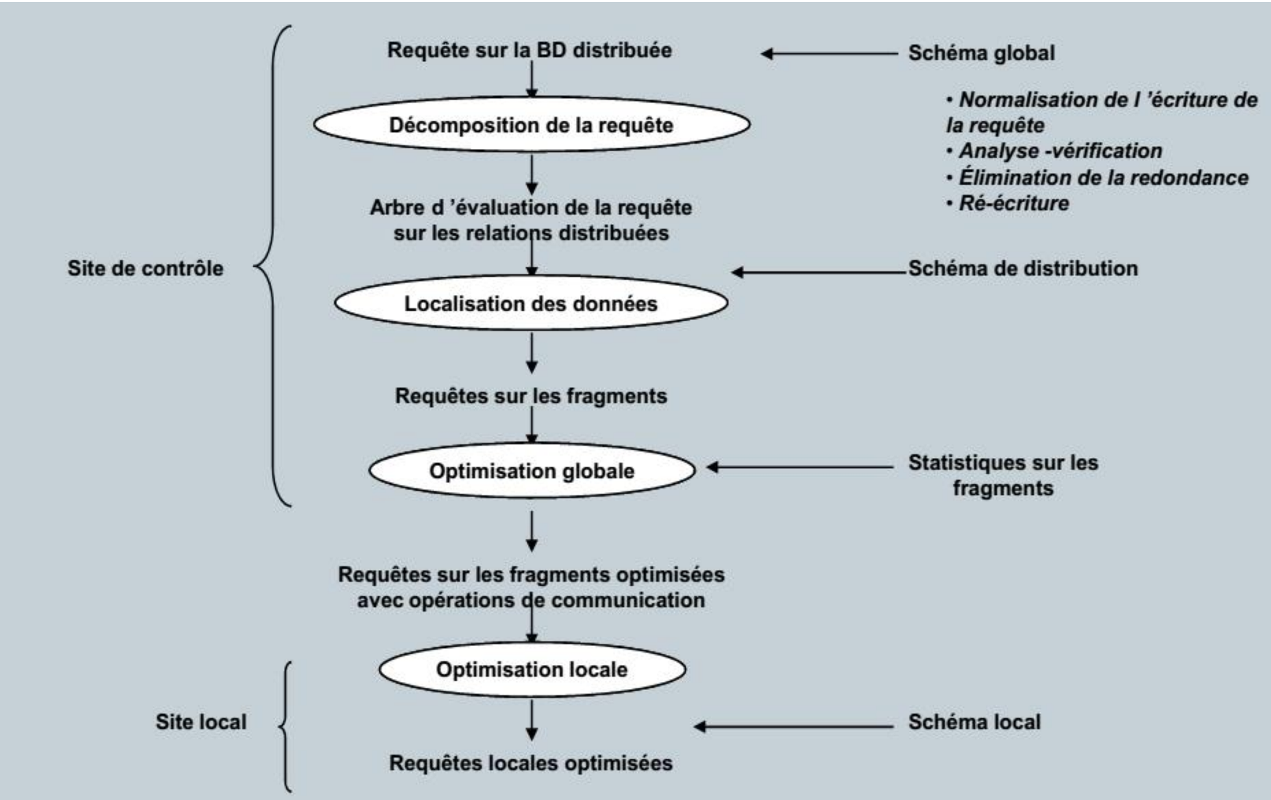


*Figure I.9.Traitement de requête distribuée **[19]**.*

Un SGBD reparti évalue une requête, écrite selon le schéma global du système, en quatre phases : optimisation logique, localisation de fragments, optimisation globale, optimisation locale et exécution.

- **Décomposition de requête :** Cette couche prend une requête exprimée en termes de relations globales et effectue une première optimisation partielle. Le résultat de cette première optimisation est un arbre algébrique relationnel fondé sur les relations globales **[19].**
- **Localisation des données :** Cette couche prend en ligne de compte la répartition des données sur le système. Elle est effectuée en remplaçant les relations globales des feuilles de l'arbre algébrique relationnel par leurs algorithmes de reconstruction (ce que l'on appelle parfois les programmes de localisation de données), c'est-à-dire les opérations algébriques relationnelles qui reconstituent les relations globales à partir des fragments constitutifs **[19].**
- **Optimisation globale** : Cette couche prend en considération des informations statistiques pour trouver un plan d'exécution proche de l'optimal. Le résultat de cette couche est une stratégie d'exécution basée sur des fragments, où viennent s'ajouter des primitives de communication qui envoient les parties de la requête aux SGBD locaux pour qu'ils les exécutent, et qui permettent ensuite de recevoir les résultats **[19].**
- **Optimisation locale :** Tandis que les trois premières couches s'exécutent sur le site de contrôle (généralement le site qu'a lancé la requête), cette couche particulière s'exécute sur chacun des sites locaux impliqués dans la requête. Chaque SGBD local effectue ses propres optimisations locales **[19].**

Les trois premières phases sont effectuées sur un site centralisé qui utilise des informations stockées dans le catalogue global du système. Cependant, la quatrième phase est effectuée sur des nœuds désignés durant la phase de l'optimisation globale.

Le catalogue global stocke les informations concernant les données partagées. Il contient des informations sur les placements des fragments et sur leurs caractères physiques. Par conséquent, ce catalogue joue un rôle essentiel dans le traitement de requêtes surtout lors de la localisation de fragments et de l'optimisation globale.

Conclusion

Les SDBDs distribuées offrent une autonomie des sites ainsi qu'une distribution de l'administration. La distribution des données entraine la révision des notions de stockage des données, du traitement des requêtes, du contrôle de l'accès simultané ainsi que de la reprise **[13]**, et nous avons vu que pour une requête on aura plusieurs stratégie d'exécution, nous allons présenter comment on peut les énumérer et choisir la meilleure stratégie dans le chapitre suivant.

Chapitre II

L'OPTIMISATION DES REQUETE DISTREBUE

Introduction

L'optimisation de requête est une opération dans laquelle plusieurs plans d'exécution d'une requête SQL sont examinés pour en sélectionner le meilleur. L'estimation de leurs coûts dépend du temps d'exécution et du nombre de ressources utilisées pour y parvenir, elle se mesure en entrées-sorties. Typiquement, les ressources coûteuses sont l'utilisation du processeur, la taille et la durée des tampons sur le disque dur, et les connexions entre les unités du parallélisme. Plusieurs SGBD, comme Oracle et MySQL, possèdent des fonctions permettant d'effectuer ces calculs via un optimiseur. Il y a deux types d'optimisation : Le plan d'exécution logique (PEL) qui dépend de l'algèbre relationnelle, et le plan d'exécution physique (PEP) qui tient compte des index et de la taille des données **[17].**

I. Les caractéristiques de l'optimiseur

I.1 Processus de l'optimisation de requête distribuée

Une requête distribuée est une requête qui choisit des données à partir des bases de données situées aux emplacements multiples dans un réseau. Le traitement de requête est beaucoup plus difficile dans l'environnement distribué que dans l'environnement centralisé, parce qu'un grand nombre de paramètres affectent l'exécution des requêtes distribuées. Le processus d'optimisation des requêtes montré dans la **figure 10** consiste à obtenir une requête sur les n relations et produire le meilleur plan d'exécution de requête (QEP).

L'optimiseur est composé par l'espace de recherche, les stratégies de recherches et de modèle de coût.

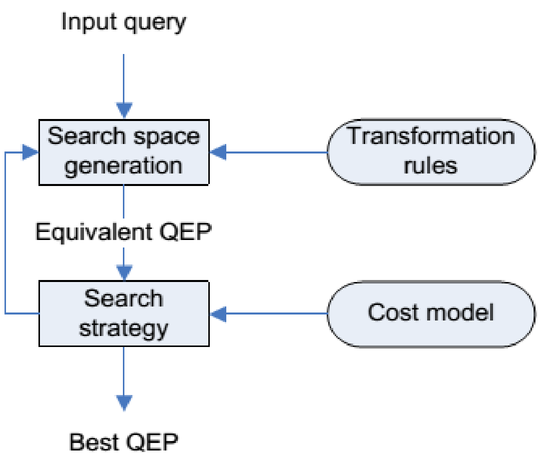


Figure II.10.Optimisation des requêtes distribuées ***[4].***

I.1.1 L'espace de recherche

L'espace de recherche, ou l'espace de solution, est l'ensemble de tous les plans d'exécutions de requête (QEPs) qui calculent le même résultat. Un point dans l'espace de solution est un plan particulier, c à d, une solution pour le problème. Dans les systèmes de base de données relationnelle, chaque plan d'exécution de requête peut être représenté par un arbre de traitement pour exécuter l'expression de jointure, où les nœuds de feuilles sont les relations de bases et les nœuds internes représentent des opérations. Différentes formes d'arbres ont été considérées : arbre gauche-profond, arbre droit-profond et arbre touffu. La figure 11 illustre les arbres de jointures pour une requête de multi-jointure R1 ∞ R2 ∞ R3 ∞ R4 **[1] [2] [3].**

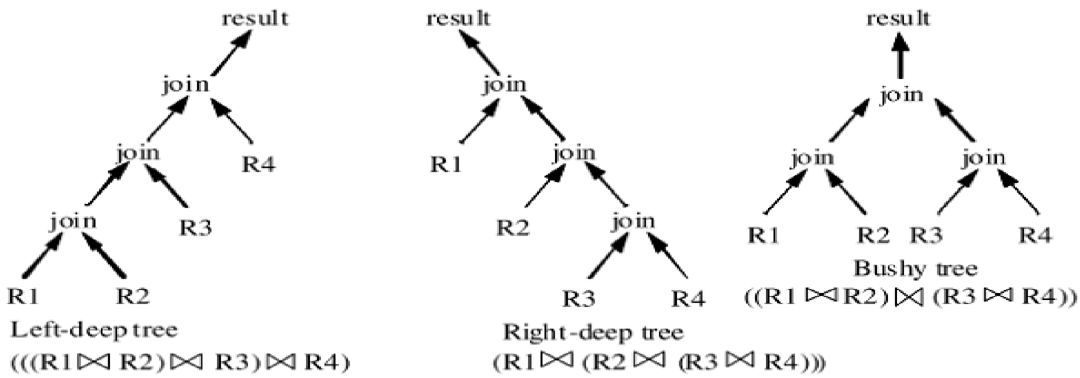


Figure II.11.Les formes d'arbres de jointure ***[3]****.*

Un espace de recherche peut être restreint selon la nature des plans d'exécution et de la stratégie de recherche appliquée. La nature des plans d'exécution est déterminée selon la forme des structures d'arbres, c à d, arbre gauche-profond, arbre droit profond et arbre touffu. Dans un environnement distribué, les arbres touffus sont utiles en exhibant le parallélisme **[3].**

I.1.2 Modèle de coût distribué

Un modèle d'optimiseur de coût inclut des fonctions de coût pour prévoir le coût des opérateurs et des formules pour évaluer les tailles des résultats. Des fonctions de coût peuvent être exprimées en ce qui concerne le temps total ou le temps de réponse. Le temps total est la somme de tous les temps, et le temps de réponse est le temps écoulé du déclenchement à l'accomplissement de la requête. Le temps total (TT) est calculé comme suit, où T_{CPU} est la période d'une instruction d'unité centrale de traitement, $T_{I/O}$ la période d'une entrée-sortie de disque, T_{MSG} la période fixe du lancement et de la réception d'un message, et TR le temps qu'il prend pour transmettre une unité de données d'un emplacement à l'autre.

TT = TCPU * #insts + TI/O * #I/Os + TMSG * #msgs + TTR * #bytes

Quand le temps de réponse de la requête est la fonction objective de l'optimiseur, le traitement local parallèle et les communications parallèles doivent également être considérés. Ce temps de réponse (RT) est calculé comme ce qui suit :

RT = TCPU * seq_#insts + TI/O * seq_#I/Os + TMSG * seq_#msgs +TTR * seq_#bytes

Les plus anciens DBMSs distribués conçu pour des réseaux WAN ont ignorés le coût de traitement local et se concentrent sur la minimisation du coût de communication. Considérer l'exemple suivant :

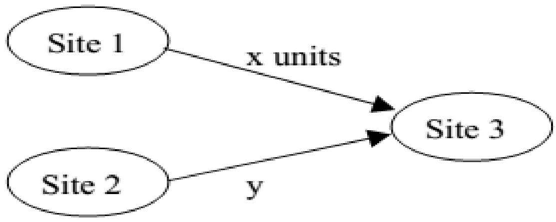


Figure II.12.Exemple d'un modèle du cout [3].

Dans le transfert parallèle, le temps de réponse est réduit au minimum en augmentant le degré d'exécution parallèle. Ceci n'implique pas que le temps total est également réduit au minimum. Au contraire, il peut augmenter le temps total, par exemple en ayant un traitement local plus parallèle (inclut souvent des frais généraux de synchronisation) et des transmissions.

I.1.3 Statistiques de base de données

Le facteur principal affectant la performance est la taille des relations intermédiaires qui sont produites pendant l'exécution. Quand une opération suivante est située à un emplacement différent, la relation intermédiaire doit être transmise au-dessus du réseau. Elle est d'intérêt principal d'estimer la taille des résultats intermédiaire afin de réduire au minimum la taille des transferts de données. L'évaluation est basée sur des informations statistiques sur les relations et les formules pour prévoir les cardinalités des résultats des opérations relationnelles.

Parfois, les données statistiques comprennent également le facteur de sélectivité de jointure pour quelques paires de relations, c'est la proportion de tuples participant à la jointure. Le facteur de sélectivité de jointure, dénotée SF_J de relations R et S est une valeur réelle comprise entre 0 et 1 :

$$SF_J(R,S) = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

- ✓ **Produit cartésien** : La cardinalité du produit cartésien de R et de S est :
- ✓ **Card(R x S) = card(R) x card(S).**
- ✓ **Jointure** : Il n'y a aucune manière générale d'estimer la cardinalité d'une jointure sans informations supplémentaires. La limite supérieure de la cardinalité de jointure est la cardinalité du produit cartésien **[3].**

I.2 Stratégies de recherche : Dans la littérature, nous distinguons généralement trois classes de stratégies d'optimisation de l'ordre de jointure qui sont les algorithmes génétiques déterministes et randomisés.

I.2.1 L'algorithme génétique

Algorithme génétique (GA) est un algorithme de recherche et d'optimisation qui suit l'évolution naturelle selon laquelle les organismes vivants s'adaptent aux changements de l'environnement. GA se compose de systèmes de codage.

Le concept de base du GAs a été développé à l'origine par Holland et révisé par la suite par Goldberg. Goldberg a montré que les gaz sont indépendants de toute hypothèse sur l'espace de recherche et sont basés sur le mécanisme de la génétique naturelle. La première étape pour modéliser ce problème comme un problème de GA consiste à déterminer le chromosome, les opérateurs de GA et la fonction de fitness **[3].**

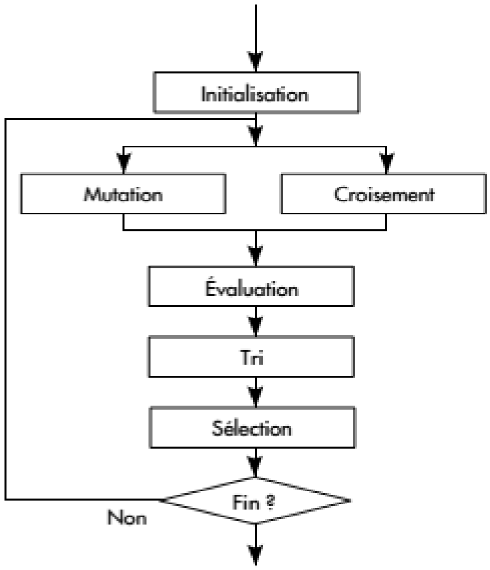


*Figure II.13.Schéma de l'algorithme Génétique **[13]**.*

Conception d'un chromosome : Afin de coder un plan d'accès différent, nous avons considéré le schéma de liste ordonnée pour représenter chaque arbre de traitement en tant qu'une liste ordonnée. Par exemple (((R1⋈R5) ⋈R3) ⋈R2) ⋈R4) est codée sous forme de "15324".

Ceci convertie l'ordre de jointure du problème de Traveling Salman Problème (TSP). Les plans de requête possible sont codés comme des chaînes entières. Chaque chaîne représente l'ordre de jointure d'une relation de la requête à l'autre. Ce mécanisme est également utilisé dans l'optimiseur de PostgreSQL. Il peut y avoir d'autres façons de coder les arbres de traitement particulièrement touffu **[3].**

Opérations de GA : Pour le croisement, un point dans le chromosome sélectionné sera sélectionné ainsi que qu'un point correspondant à un autre chromosome et ensuite les queues seraient échangées.

Chaque chromosome représente une stratégie de gauche profonde. Le chromosome est une liste ordonnée de gènes, et chaque gène consiste en une relation et une méthode de jointure. L’exemple suivant représente la stratégie de gauche profonde **[15]**.

$^{J}A \quad ^{J}C \quad ^{J}B \quad ^{J}D \quad ^{J}F \quad ^{J}E$

La Fonction Fitness :

Définir la fonction fitness est l'une des étapes les plus importantes dans la conception d'une méthode axée sur les GA qui peut guider la recherche vers la meilleure solution.

Conception de l'algorithme :

Après le calcul de la fonction fitness pour chaque chromosome parent, l'algorithme générera N enfants. La Faible valeur de la fonction fitness d'un chromosome parent est la plus forte probabilité, qu'il doit contribuer à la génération d’un ou plusieurs descendants dans la prochaine génération. Après avoir effectué des opérations, certains chromosomes ne pourraient pas satisfaire la fitness et en conséquence l'algorithme élimine ce processus et obtiendra M ($M \leq N$) enfants chromosomes. L'algorithme sélectionne ensuite les chromosomes N avec la valeur basse de la fonction fitness de M + N chromosomes (M enfants et N parents) pour être parent des prochaines générations. Ce processus se répète jusqu'à ce qu'un certain nombre de générations soient traité, après le meilleur chromosome est choisi **[3].**

I.2.2 Stratégies déterministes

Également connue comme une recherche exhaustive, pour une requête donnée, l'ensemble de tous les plans possibles d'exécution est énuméré, elles établissent des plans d'exécution à partir des sous-palan déjà optimisés en commençant par une partie ou l'ensemble des relations de base d'une requête. Des plans partiels équivalents sont construits, et aussi comparés sur un certain modèle de coût. Pour réduire le coût d'optimisation, des plans partiels qui ne sont pas susceptibles de mener au plan optimal sont élagués aussitôt que possible, et les meilleurs plans sont employés pour construire le plan complet. Les stratégies déterministes de base qui sont la programmation dynamique, l’algorithme avide et l’approche de combinaison des deux algorithmes seront présentées **[2] [3] [9].**

I.2.2.1 Programmation dynamique

La programmation dynamique est une méthode d'optimisation des processus de décisions séquentielles .elle s'appuie sur l'algorithme de Bellman, dont la traduction sur le graphe associe au processus permet de rechercher le chemin de valeur optimale .les programmes les plus simples sont ceux ou l'avenir est détermine : l’intérêt de l'algorithme est évident quand le caractère combinatoire du problème amène à comparer un très grand nombre de politiques. Mais dans beaucoup de cas concrets l'avenir est incertain. Dans le cas où il est probabilisable, il est possible de comparer les stratégies en les évaluant d'après leur espérance mathématique.

I.2.2.2 Stratégies randomisées

Les algorithmes randomisés effectuent habituellement les marches aléatoires dans l'espace de solution par l'intermédiaire d'une série de mouvements. Les genres de mouvements qui sont considérés dépendent de l'espace de solution. Si des arbres de traitement gauche profonds sont employés, chaque solution peut être représentée uniquement par une liste numérotée de relations participantes à la jointure. Il y a différents mouvements pour modifier ces relations, swap et 3 Cycle. Si c'est les arbres de traitement touffus, les mouvements suivants sont employés, le choix de la méthode de jointure, jointure associative/commutative. Les stratégies aléatoires commencent généralement par un premier plan d'exécution qui est itérativement amélioré par l'application d'un ensemble de règles de transformation (mouvements). Les plans initiaux peuvent être obtenus par une stratégie déterministe, deux techniques d'optimisation ont été abondamment déjà étudiées et comparées : l'amélioration itérative et le recuit simulé **[6] [7] [9].**

- **Amélioration itérative**

L'algorithme a une boucle intérieure qui s'appelle l'optimisation locale, qui commence par un point aléatoire et améliore la solution en acceptant à plusieurs reprises des mouvements inclinés aléatoires jusqu'à ce qu'elle atteigne le minimum local. L'amélioration itérative répète ces optimisations locales jusqu'à ce qu'une condition d'arrêt (un nombre prédéterminé de points de départ sont traités ou un délai est dépassé) soit remplie, à tel point, il renvoie le minimum local avec le coût minimal **[2].**

- **Recuit simulé**

Le recuit simulé est une amélioration de l'algorithme d'amélioration itérative, il accepte des mouvements ascendants avec une certaine probabilité, essayant d'éviter d'être attrapée dans un minimum local avec un coût élevé. Il a été à l'origine dérivé par analogie pour traiter le recuit des cristaux. L'algorithme à une boucle intérieure qui s'appelle cycle. Chaque cycle est exécutée sous une valeur fixe d'un paramètre T, appelée la température, qui contrôle la probabilité d'accepter des mouvements ascendants **[7] [10].**

- **Principe du recuit simulé**

Explorer l'espace d'état de manière aléatoire afin d'éviter les minimums locaux.
Diminuer progressivement la température T(n) pour stabiliser l'algorithme sur un minimum global.
Si le refroidissement est trop rapide, il y a un risque de rester bloqué dans un minimum local (configuration sous-optimale).

- **Les domaines d'application de recuit simulé**

- La CAO (conception de circuits, placement de composants).
- Le traitement d'images (restitution d'images brouillées).

- Le routage des paquets dans les réseaux.
- Le problème du voyageur de commerce.
- Le problème du sac à dos.

➤ **Avantages et inconvénients de recuit simulé**

❖ **Les avantages :**

- Facile à implémenter.
- Donne généralement de bonnes solutions par rapport aux algorithmes de recherche classiques.
- Peut être utilisé dans la plupart des problèmes d'optimisation.

❖ **Les inconvénients :**

- Non-convergence vers l'optimum peut se rencontrer assez vite.
- L'impossibilité de savoir si la solution trouvée est optimale.
- La difficulté de déterminer la température initiale.
- Très coûteuse en temps de calcul.

II. Les Travaux Relatifs

Parmi les travaux réalisés dans le domaine de l'optimisation des requêtes distribuées, nous citons les suivants :

II.1 Optimisation de requête en utilisant les systèmes Multi Agent

Dans ce travail **[4]**, ils ont proposé une nouvelle architecture de système multi-agent basée sur frameJava pour traiter une requête distribuée, une requête est soumise par l'utilisateur à l'agent de distributeur de requête, puis elle sera distribuée. Leur architecture proposée emploie différents types d'agents, chacun à ses propres caractéristiques. L'architecture de système proposée est montrée dans la figure suivante :

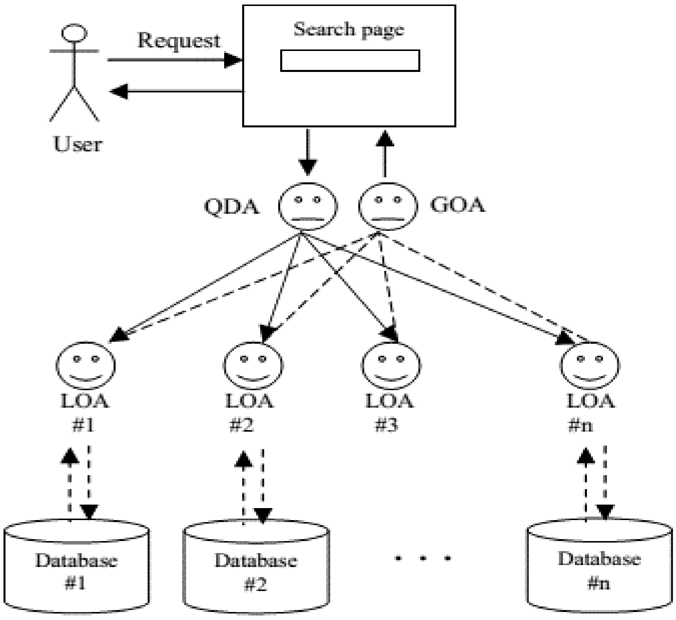


Figure II.14.Système de l'architecture proposée ***[4]****.*

- **Agent de distributeur de requête (QDA) :** Après la réception de la requête d'utilisateur, le QDA envoie des sous-requêtes aux agents optimiseurs locaux. Le QDA peut également créer des agents de recherche si nécessaire **[4]**.
- **Agents optimiseur locaux (LOAs) :** Ces agents appliquent une optimisation de sous-requête basée sur l'algorithme génétique, et renvoient la taille de la table du résultat à l'agent optimiseur global [**4**].
- **Agent optimiseur global (GOA) :** Cet agent a responsabilité de trouver le meilleur d'ordre de jointure via le réseau. Pour faire ainsi, GOA reçoit l'information de taille de la table du résultat de LOAs et suit, encore, une méthode évolutionnaire afin de trouver un ordre de joindre semi-optimal. Cependant, cette fois la fonction fitness (coût) de GA est basée sur la minimisation de communication parmi différents emplacements. Le résultat final serait alors envoyé à LOAs pour effectuer l'opération et pour fournir le résultat au GOA pour délivrer le résultat au client **[4]**.

II.2 Stratégies d'optimisation de requêtes pour les bases de données distribuées

Il a présenté le flux de l'optimisation des requêtes composée de plusieurs modules pour les bases de données distribuées. Le module utilisateurs dans les systèmes distribués analyse la demande de requête de l'utilisateur. Le module d'analyse système examine la phrase de la requête, où sa sémantique et la syntaxe et sorts sont vérifiés puis en convertissant la requête dans son arborescence correspondante. **[8]**

On passe Cet arbre correspondant est transmis au module de conversion requête arbre qui la convertit dans l'arborescence de requêtes globale selon la structure de données décrite dans l'arborescence de la requête. L'arborescence de requêtes globale provenant du module de conversion requête arbre est mappée aux arbres les opérateurs physiques correspondants par le module de l'optimiseur qui sélectionne ensuite un arbre de l'opérateur physique avec le plus bas coût **[8]**.

Le dictionnaire de données local est ajouté avec la table de la phrase pour stocker des résultats le plus souvent utilisé pour éviter la transmission de données de grande taille, ce qui améliore grandement l'efficacité des requêtes. Mais lorsque la taille de la table correspondante devenu grand, le temps de traitement du processeur sera grand et la consommation de mémoire sera plus qui peut être envisagé la limitation du système **[8]**.

II.3 Des agents mobiles à l'optimisation de requêtes réparties

Dans un contexte de bases de données réparties à grande échelle, l'exécution d'un plan engendré par un optimiseur classique peut présenter des mauvaises performances à cause de la centralisation des décisions prises par l'optimiseur, de l'imprécision des estimations, et de l'indisponibilité des ressources **[5].**

Les statistiques stockées dans les catalogues système, relatives aux données de base, sont sujettes principalement à l'obsolescence. En effet, dans un réseau à grande échelle, il est très difficile de maintenir continuellement à jour les statistiques concernant les relations de base distantes. Les estimations qui reposent sur celles-ci, de la sélectivité des prédicats (de jointure et de sélection) à la taille des résultats intermédiaires, ne peuvent alors être que plus imprécises. De plus, les estimations relatives à la disponibilité des ressources en termes de la charge des processeurs, de la taille mémoire disponible, et de la bande passante en E/S et en réseau ne peuvent être que fausses **[5].**

II.3.1 Modèle d'exécution

Dans l'objectif de concevoir et de développer un modèle d'exécution pour des requêtes réparties mobiles, il devient donc opportun de rendre, d'abord, autonome chaque agent réalisant un opérateur relationnel (jointure), conscient de son environnement, et capable de décider de l'endroit pour continuer son exécution sur le site destination. Cet opérateur mobile constitue la brique de base pour une exécution autonome et adaptable d'une requête. Dans cette perspective, ils ont défini une politique de déplacement des agents mobiles issus des opérateurs de requêtes réparties soumises au système en réagissant aux imprécisions des estimations. Avec la mobilité logicielle, les algorithmes de jointure directe et à base de semi-jointure ont été étendus afin de décentraliser la décision et le contrôle. Ce n'est plus l'optimiseur qui choisit où seront exécutées les jointures, mais c'est la jointure elle-même, qui prend la décision de se déplacer ou non. En effet, l'agent exécutant une jointure s'adapte à l'évolution de l'environnement système et réagit aux imprécisions des estimations dues au profil de l'application (taille des relations intermédiaires) calculées à la compilation et en tenant compte des caractéristiques des ressources systèmes **[5].**

II.3.2 Travaux en cours : Les résultats obtenus relatifs à l'apport des agents mobiles à l'optimisation des opérateurs relationnels sont considérés comme des optimisations locales. Ce travail nécessite donc d'être enrichi et étendu par : l’étude de l'apport des agents mobiles à deux niveaux :

Niveau 1 :

- Dans l'optimisation dynamique des requêtes entières, notamment, en définissant une stratégie pour les liens de poursuite qui acheminent les relations temporaires vers les opérateurs qui se sont déplacés. Cette stratégie devrait éviter la recopie des messages pour ne pas détériorer les performances en terme de temps de réponse.

- Dans le processus de décentralisation du contrôle effectué par l'optimiseur évitant ainsi un goulet d'étranglement.

Niveau 2 : L’évaluation des surcoûts dus au contrôle décentralisé et la détermination de son impact sur les performances.

II.4 Optimisation des requêtes dans des bases de données distribuées

L’optimisation des requêtes est d'exploiter la requête différemment afin qu'il donne le même résultat, mais augmenté la vitesse pour récupérer les données. Les requêtes devraient être efficaces afin que les données puissent être récupérées. Il existe des méthodes alternatives pour exécuter la requête qui donnent le même résultat **[16]**.

Dans les systèmes distribués, le coût d'un plan de requête est donné par la somme de coût de transmission et le coût de transformation locale. Le coût de transmission est en facteurs de vitesse pour transférer les données d'une machine à une autre machine et le coût de traitement local est en termes de cycles de processeur, E/S disque **[16]**.

L’optimiseur de requête, détermine :
- Le nombre des plans alternatifs.
- Le cout de chaque plan utilisé.
- Le choix du plan avec le meilleur coût.

Pour l’Optimisation de la requête de jointure dans les bases de données centralisées est simple par rapport aux bases de données distribuées **[16]**.

II.4.1 Travaux connexes sur l'optimisation des requêtes de jointure dans la base de données répartie

Il existe différentes méthodes pour optimiser et améliorer les performances des requêtes dans les bases de données.

L'optimisation des requêtes de jointure dans les bases de données distribuées consiste à calculer la taille des données sur deux machines différentes, et ensuite d'envoyer le tableau ayant une taille inférieure à l'autre site et puis exécuter la requête de jointure **[16]**.

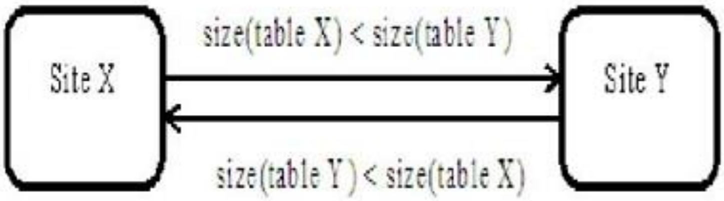


*Figure II.15.Réduire la taille des données de transmission **[16]**.*

Une autre méthode de requête de jointure dans les bases de données distribuées est le traitement de requête parallèle. Le traitement parallèle ne se concentre pas sur réduire la quantité de données de transmission, mais plutôt de maximiser le nombre de transmissions simultanées **[16]**.

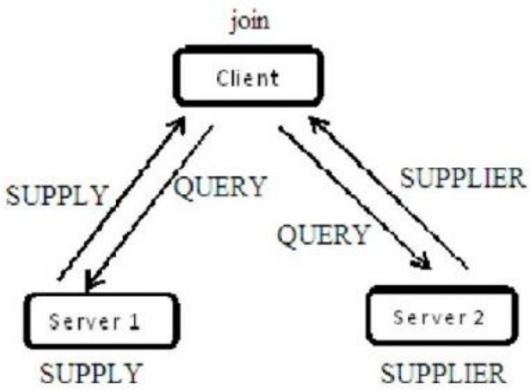


*Figure II.16.Traitement parallèle de la requête de jointure **[16]**.*

Le client envoie la requête au serveur 1 et serveur 2. Après cela, le serveur 1 envoie les données SUPLLY et serveur 2 envoie les données SUPPLIER au client. Puis le client insère les données dans sa base de données et effectue la requête de jointure des données provenant de deux serveurs.

II.4.2 Objectifs de la requête de jointure dans les bases de données distribuées : Il existe plusieurs objectives :

- **Taille des données transmises :** c'est la quantité de données qui doit être transmise.
- **Vitesse de Transmission :** elle dépend de la vitesse du réseau.
- **Les coûts de traitement local** : Il est composé par le coût de l'CPU et le coût E/S.

Conclusion

Pour conclure, d'optimisation de requête permettent sélectionner le meilleur. L'estimation de leurs coûts dépend du temps d'exécution et du nombre de ressources utilisées et les ressources coûteuses sont l'utilisation du processeur, la taille et la durée des tampons sur le disque dur, on vu aussi tout d’abord nous présentons la complexité de traitement de requête dans cet environnement et abordons le problème de l’optimisation de requête distribuée, et finalement, nous présentons les travaux relatifs à ce domaine.

Chapitre III

L'APPROCHE PROPOSE

Introduction

Parmi tous les types d'algorithmes existants, certains ont la particularité de s'inspirer de l'évolution des espèces dans leur cadre naturel. Ce sont les algorithmes génétiques. Les espèces s'adaptent à leur cadre de vie qui peut évoluer, les individus de chaque espèce se reproduisent, créant ainsi de nouveaux individus, certains subissent des modifications de leur ADN, certains disparaissent… .

Un algorithme génétique va faire évoluer une population dans le but d'en améliorer les individus. Et c'est donc, à chaque génération, un ensemble d'individus qui sera mis en avant et non un individu particulier. Nous obtiendrons donc un ensemble de solutions pour un problème et pas une solution unique. Les solutions trouvées seront généralement différentes, mais seront d'une qualité équivalente. Nous reviendrons sur cette notion de qualité des solutions dans la partie 2 (L'évaluation des individus).

On peut détailler le déroulement d'un algorithme génétique dans la figure suivant :

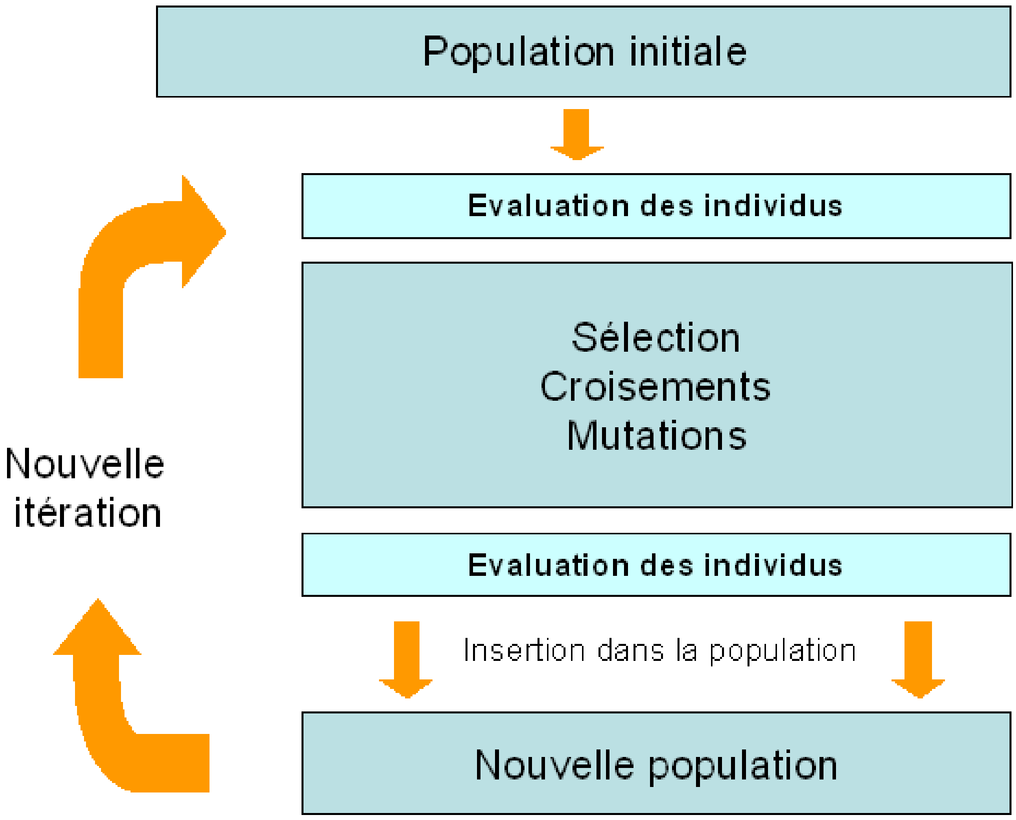


Figure III.17 : Le déroulement d'un algorithme génétique.

I. Algorithme génétique

I.1 La création de la population initiale

Pour démarrer un algorithme génétique, il faut lui fournir une population à faire évoluer. La manière dont le programmeur va créer chacun des individus de cette population est entièrement libre. Il suffit que tous les individus créés soient de la forme d'une solution potentielle, et il n'est nullement besoin de songer à créer des bons individus. Ils doivent juste rentrer dans le 'moule' du problème posé.

Par exemple, si vous cherchez un chemin entre 2 points, les individus créés devront simplement posséder le bon point de départ et le bon point d'arrivée, peu importe par où ils passent. De même si vous cherchez des solutions pour un jeu d'échecs, il suffira que les individus créés possèdent des mouvements autorisés sur de pièces existantes. Même si les individus créés n'ont aucune chance d'être des solutions acceptables pour le problème posé, ils peuvent en avoir la forme. Il n'y a donc aucune objection à les mettre dans la population initiale.

Il est tout à fait possible de créer les individus de manière aléatoire. Et cette méthode amène un concept très utile dans les algorithmes génétiques : la diversité. Plus les individus de la population de départ seront différents les uns des autres, plus nous aurons de chance d'y trouver, non pas la solution parfaite, mais de quoi fabriquer les meilleures solutions possibles.

I.2 Et le hasard dans tout ça ?

Il est tout à fait possible de créer les individus de manière aléatoire. Et cette méthode amène un concept très utile dans les algorithmes génétiques : la diversité. Plus les individus de la population de départ seront différents les uns des autres, plus nous aurons de chance d'y trouver, non pas la solution parfaite, mais de quoi fabriquer les meilleures solutions possibles. Et pour détailler cette aidé, voilà l'exemple qui parle-t-on le problème d'informaticien suivant :

I.3 Problème de l'informaticien en vacances

Nous n'allons pas parcourir tous les chemins possibles, il y en aurait trop. Nous créons une population d'individus au hasard, par exemple

{A,B,C,D,E,F,G,H,I,J},{D,A,F,J,C,E,G,H,B,I},{J,A,H,F,D,C,I,G,E,B},{J,F,A,C,B,E,I,H,G,D }. Une population intéressante pourrait être de taille 100, l'essentiel étant que les individus contiennent toutes les villes une et une seule fois, et qu'ils soient relativement différents les uns des autres.

I.4 La taille de la population à manipuler

La taille de la population initiale est également laissée à l'appréciation du programmeur. Il n'est souvent pas nécessaire d'utiliser des populations démesurées. Une taille de 100 ou 150 individus s'avèrera souvent amplement suffisante, tant pour la qualité des solutions trouvées que pour le temps d'exécution de notre algorithme. Evidemment, ce n'est qu'un ordre de grandeur. Ensuite, libre à vous de modifier la taille de la population initiale en fonction du problème à résoudre si les solutions trouvées ne vous conviennent pas.

I.5 L'évaluation des individus

Une fois que la population initiale a été créée, nous allons en sortir les individus les plus prometteurs, ceux qui vont participer à l'amélioration de notre population. Nous allons donc attribuer une 'note' ou un indice de qualité à chacun de nos individus. La méthode d'évaluation des individus est laissée au programmeur en fonction du problème qu'il a à optimiser ou à résoudre.

Cette étape intermédiaire d'évaluation peut même devenir une étape importante du processus d'amélioration de notre population. En effet, les différents individus ne sont pas toujours comparables, il n'est pas toujours possible de dire qu'un individu est meilleur ou moins bon qu'un autre. C'est le cas des problèmes multicritères, lorsqu'une solution dépend de plusieurs paramètres. Vous pourrez toujours dire qu'un nombre est supérieur ou non à un autre, mais vous ne pourrez pas dire si un vecteur est supérieur ou non à un autre. La notion de supériorité pour les vecteurs n'existe pas. Vous pouvez comparer leur norme, mais pas les vecteurs eux-mêmes.

I.6 La création de nouveaux individus

I.6.1 La sélection

Comme son nom l'indique, la sélection vise à sélectionner une sous population à partir d'une population parent. La méthode la plus courante est celle initiée par Holland lui-même en 1975 : la "roulette wheel" sélection, qui est une méthode de sélection proportionnelle au niveau de fitness des individus. Le nombre de fois qu'un individu sera sélectionné est égal à son fitness divisé par la moyenne du fitness de la population totale (plus exactement, la partie entière représente le nombre de fois qu'il sera sélectionné, et la partie flottante la probabilité qu'il aura d'être sélectionné à nouveau).

Il est tout à fait possible de choisir des individus au hasard et de les mélanger aléatoirement pour créer de nouveaux individus. Nous allons détailler ici quelques-unes de méthodes de sélections souvent utilisées : la roulette, la sélection par rang, la sélection par tournoi et l'élitisme.

I.6.1.1 La roulette

La sélection des individus par le système de la roulette s'inspire des roues de loterie. A chacun des individus de la population est associé un secteur d'une roue. L'angle du secteur étant proportionnel à la qualité de l'individu qu'il représente. Vous tournez la roue et vous obtenez un individu. Les tirages des individus sont ainsi pondérés par leur qualité. Et presque logiquement, les meilleurs individus ont plus de chance d'être croisés et de participer à l'amélioration de notre population.

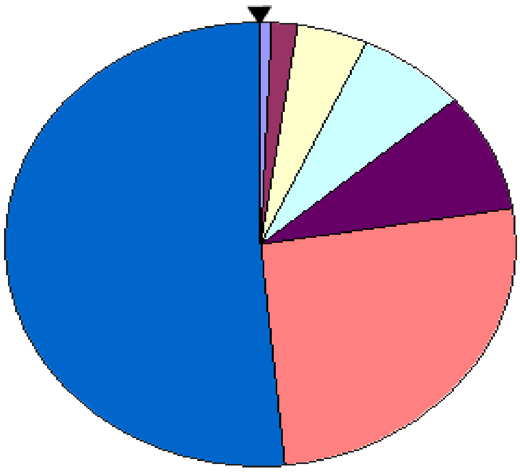


Figure III.18: Schéma d'une sélection roulette.

I.6.1.2. La sélection par rang

La sélection par rang est une variante du système de roulette. Il s'agit également d'implémenter une roulette, mais cette fois ci les secteurs de la roue ne sont plus proportionnels à la qualité des individus, mais à leur rang dans la population triée en fonction de la qualité des individus.

D'une manière plus parlante, il faut trier la population en fonction de la qualité des individus puis leur attribuer à chacun un rang. Les individus de moins bonne qualité obtiennent un rang faible (à partir La création de la population initiale). Et ainsi en itérant sur chaque individu on finit par attribuer le rang N au meilleur individu (où N est la taille de la population). La suite de la méthode consiste uniquement en l'implémentation d'une roulette basée sur les rangs des individus. L'angle de chaque secteur de la roue sera proportionnel au rang de l'individu qu'il représente.

Le tableau suivant fournit un exemple de sélection par rang. Avec cette méthode de sélection, tous les chromosomes ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution. Ceci est dû au fait que les meilleurs chromosomes ne diffèrent pas énormément des plus mauvais.

Chromosomes	1	2	3	4	5	6	Total
Probabilités initiales	89 %	5 %	1 %	4 %	3 %	2 %	100 %
Rang	6	5	1	4	3	2	21
Probabilités finales	29 %	24 %	5 %	19 %	14 %	9 %	9 %

Tableau III.1 : Exemples de sélection par rang pour 6 chromosomes.

I.6.1.3 La sélection par tournoi

Le principe de la sélection par tournoi augmente les chances pour les individus de piètre qualité de participer à l'amélioration de la population. Le principe est très rapide à implémenter. Un tournoi consiste en une rencontre entre plusieurs individus pris au hasard dans la population. Le vainqueur du tournoi est l'individu de meilleure qualité. Vous pouvez choisir de ne conserver que le vainqueur comme vous pouvez choisir de conserver les 2 meilleurs individus ou les 3 meilleurs. A vous de voir, selon que vous souhaitez créer beaucoup de tournois, ou bien créer des tournois avec beaucoup de participants ou bien mettre en avant ceux qui gagnent les tournois haut la main. Vous pouvez faire participer un même individu à plusieurs tournois. Une fois de plus, vous êtes totalement libre quant à la manière d'implémenter cette technique de sélection.

I.6.1.4 L'élitisme

Cette méthode de sélection permet de mettre en avant les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui vont participer à l'amélioration de notre population. Cette méthode a l'avantage de permettre une convergence (plus) rapide des solutions, mais au détriment de la diversité des individus. On prend en effet le risque d'écarter des individus de piètre qualité, mais qui aurait pu apporter de quoi créer de très bonnes solutions dans les générations suivantes.

Une autre possibilité relevant aussi du domaine de l'élitisme, pour s'assurer que les meilleurs individus feront effectivement partie de la prochaine génération, est tout simplement de les sauvegarder pour pouvoir les rajouter à coup sûr dans la population suivante (en étape 4 : l'insertion des nouveaux individus dans la population.

Le nombre d'individus que vous allez sélectionner en vue d'un croisement ou d'une mutation est encore une fois laissé à votre appréciation. Pensez juste qu'il n'est pas nécessaire (et pas recommandé non plus) de modifier tous les individus d'une population

I.6.1.5 Sélection Steady-State

Ce n'est pas une méthode particulière de sélection des chromosomes parents. L'idée principale est qu'une grande partie de la population puisse survivre à la prochaine génération. L'algorithme génétique marche alors de la manière suivante. A chaque génération sont sélectionnés quelques chromosomes (parmi ceux qui ont le meilleur coût) pour créer des chromosomes fils. Ensuite les chromosomes les plus mauvais sont retirés et remplacés par les nouveaux. Le reste de la population survie à la nouvelle génération.

I.6.2 Les croisements

Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants. Initialement, le croisement associé au codage par chaînes de bits est le croisement à découpage de chromosomes (slicing crossover). Pour effectuer ce type de croisement sur des chromosomes constitués de M gènes, on tire aléatoirement une position dans chacun des parents. On échange ensuite les deux sous-chaînes terminales de chacun des deux chromosomes, ce qui produit deux enfants C_1 et C_2 (voir figure suivant).

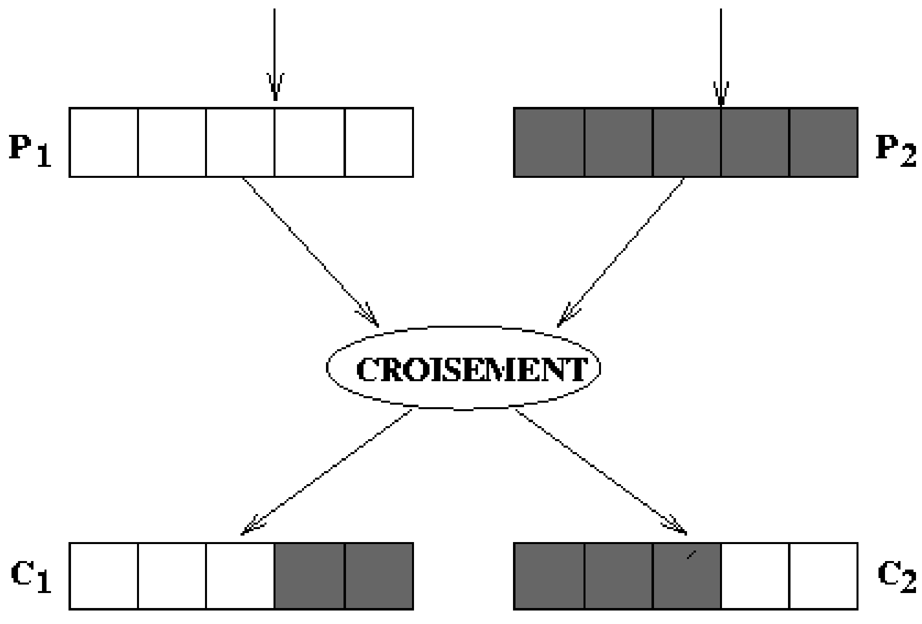


Figure III.19: Slicing crossover « tranchage croisé »

On peut étendre ce principe en découpant le chromosome non pas en 2 sous-chaînes mais en 3, 4, etc., (voir figure suivant).

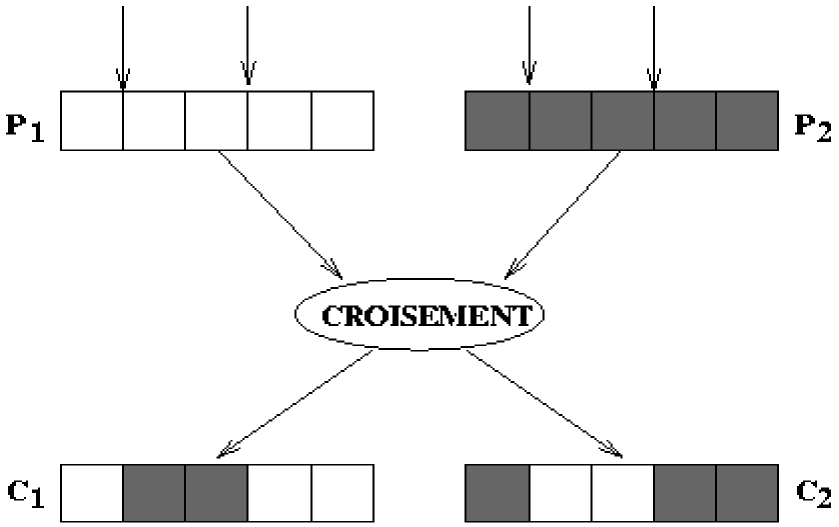


Figure III.20: Slicing crossover à 2 points. « Tranchage croisé à deux points»

Pour le taux de croisement est en général assez fort et se situe entre 70% et 95% de la population totale.

I.6.2.1 Les croisements multipoints

Il faut découper en N (2 ou 3 peuvent suffire) morceaux chacun des individus choisis, puis il faut prendre un gène de chaque individu pour créer un nouvel individu. Et il est possible de créer ainsi plusieurs individus : si un gène d'un individu ne sert pas à la création d'un individu, il peut servir à la création d'un autre.

Donc en prenant X individus à croiser, vous pouvez potentiellement obtenir X nouveaux individus. Mais rien ne vous empêche d'utiliser plusieurs fois certains gènes, pour obtenir plus de X nouveaux individus.

Aussi qu'il n'est pas nécessaire et surtout pas recommandé de croiser tous les individus d'une population, car rien ne nous dit si le résultat d'un croisement sera meilleur ou moins bon que les individus parents.

Individus parents			
Gène1	Gène2	Gène3	Gène4
Gène1	Gène2	Gène3	Gène4
Gène1	Gène2	Gène3	Gène4

Individus enfants			
Gène1	Gène2	Gène3	Gène4
Gène1	Gène2	Gène3	Gène4
Gène1	Gène2	Gène3	Gène4

Figure III.21: Exemple de croisement multipoints

I.6.3. Les mutations

Une autre solution que le croisement pour créer de nouveaux individus est de modifier ceux déjà existants. Une fois de plus, le hasard va nous être d'une grande utilité. Il peut s'avérer efficace de modifier aléatoirement quelques individus de notre population en en modifiant un gène ou un autre. Rien ne nous dit que l'individu muté sera meilleur ou moins bon, mais il apportera des possibilités supplémentaires qui pourraient bien être utiles pour la création de bonnes solutions. De même que pour les croisements, il n'est pas recommandé de faire muter tous les individus. Il est possible de faire muter un individu de la manière qu'il vous plaira. Une seule contrainte : l'individu muté doit être de la forme d'une solution potentielle.

Généralement, on ne modifie qu'un gène pour passer d'une solution à une autre solution de forme similaire mais qui peut avoir une évaluation totalement différente.

A la différence du croissement le taux de mutation est généralement faible et se situe entre 0.5% et 1% de la population totale. Ce taux faible permet d'éviter une dispersion aléatoire de la population et n'entraine que quelques modifications sur un nombre limité d'individus.

I.6.4. L'insertion des nouveaux individus dans la population

Une fois que nous avons créé de nouveaux individus que ce soit par croisements ou par mutations, il nous faut sélectionner ceux qui vont continuer à participer à l'amélioration de notre population. Une fois encore, libre au programmeur de choisir ceux qu'il souhaite conserver. Il est possible de refaire une étape d'évaluation des individus nouvellement créés. De même qu'il est possible de conserver tous les nouveaux individus en plus de notre population.

Il n'est toutefois pas recommandé de ne conserver que les nouveaux individus et d'oublier la population de travail. En effet, rien ne nous dit que les nouveaux individus sont meilleurs que les individus de départ.

Une méthode relativement efficace consiste à insérer les nouveaux individus dans la population, à trier cette population selon l'évaluation de ses membres, et à ne conserver que les N meilleurs individus.

I.6.5. Comment choisir le nombre N d'individus à conserver ?

Le nombre d'individus N à conserver est à choisir avec soin. En prenant un N trop faible, la prochaine itération de l'algorithme se fera avec une population plus petite et elle deviendra de plus en plus petite au fil des générations, elle pourrait même disparaître. En prenant un N de plus en plus grand, nous prenons le risque de voir exploser le temps de traitement puisque la population de chaque génération sera plus grande.

Une méthode efficace est de toujours garder la même taille de population d'une génération à l'autre, ainsi il est possible de dérouler l'algorithme sur un grand nombre de générations.

I.6.6 Et on passe à la génération suivante

Une fois la nouvelle population obtenue, vous pouvez recommencer le processus d'amélioration des individus pour obtenir une nouvelle population et ainsi de suite ...

I.6.7 Réitération du processus

Le programmeur a l'opportunité d'évaluer les individus de sa population avant et/ou après les phases de création d'individus. En effet, il peut s'avérer pertinent de les évaluer avant de les insérer dans la future population, de même qu'il peut être utile de les réévaluer au début de la génération suivante, si par exemple la méthode d'évaluation dépend de la taille de la population (qui a très bien pu changer). Ainsi on peut être amené à évaluer deux fois par génération chacun des individus.

Le nombre de génération est aussi laissé à l'appréciation du programmeur. Généralement il n'est pas possible de trouver des solutions convenables en moins de 10 générations et au bout de 500 générations, les solutions n'évoluent plus. Mais ceci n'est qu'un ordre de grandeur, tout dépend du problème à résoudre.

Une fois le nombre maximum de générations atteint, vous obtenez une population de solutions. Mais rien ne vous dit que la solution théorique optimale aura été trouvée. Les solutions se rapprochent des bonnes solutions, mais sans plus. Ce n'est pas une méthode exacte.

I.6.8 Les domaines d'applications de l'algorithme génétique

- La reconnaissance des formes.
- Problème de Voyageur de Commerce (PVC).
- Problème Coloration de Graphes.
- Problèmes d'ordonnancement

I.6.9 Avantage et Inconvénient de l'algorithme génétique

- ✓ **Avantage :**
- ➢ La gestion efficace d'une population de solutions
- ➢ L'ajout d'une recherche locale à un algorithme génétique peut compenser la faiblesse des AG en vitesse de convergence qui est très lente.
- ✓ **Inconvénient :**

Il est difficile pour parcourir efficacement l'espace des solutions souvent très vaste.

II. Approche Proposé

II.1. Algorithme génétique

Dans notre cas, une population sera un ensemble d'individus, un individu sera une solution à un problème donné (optimisation de requête distribuée), un gène (un individu) sera une partie d'une solution, donc d'un individu, une génération est une itération de notre algorithme.

II.1.1 La création de la population initiale

La création de la population initiale pour chaque requête donnée, cette population c'est ensemble d'individus généré depuis la table des placements (à partir de la base de données) par exemple on a la table des placements suivant :

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
R1							
R2							
R3							
R4							

Tableau III.2: Table des placements.

La table de placement représente l'emplacement des relations dans les sites, alors dans notre cas la population initial c'est le résultat de la requête déjà sélectionnée par l'utilisateur.

Pour détailler l aidé précède on a l'exemple suivant :

Select * **From** R1, R3, R4 **Where** « condition ».

Les plans d'exécution de cette requête on examine la clause From et on vérifie dans table de placement les sites appropriés par exemple on aura :

Le 1^{er} chromosome « c'est le 1^{er} individu dans la population initial » :

S1	S2	S3	S1

Le 2eme chromosome « c'est le 2^{eme} individu dans la population initial » :

S1	S2	S3	S1

II.1.2 La taille de la population à manipuler

Généralement la taille de la population initiale est également laissée à l'appréciation du programmeur. Dans notre cas on a fixé en 20 individus avec la possibilité de changement comme paramètre d'entré par l'utilisateur.

II.1.3 L'évaluation des individus

Apre la création de la population initiale, nous allons en sortir les individus les plus prometteurs, ceux qui vont participer à l'amélioration de notre population à partir de la fonction de fitness suivant : TMSG * #msgs + TTR * #bytes, avec le TMSG c'est le tempe d'un message et #msgs c'est le paquet (la taille des résultats à transférer) suivant le contrainte de réseau et TTR c'est temp total de response d'un message #bytes c'est la taille de paquet (message) en octets. Dans notre cas le modèle de cout c'est cout de communication.

Apre l'application de la fonction de fitness précède le résultat sera dans l'ordre croissant.et on choisit les meuliers 20 individus comme une population initial.

II.1.4 La création de nouveaux individus

II.1.4.1 La sélection

Comme mentionné précédemment, il existe plusieurs méthodes de sélections souvent utilisées : la roulette, la sélection par rang, la sélection par tournoi et l'élitisme.

Dans notre cas nous utilisé la sélection par l'élitisme parce que cette méthode de sélection permet de mettre en avant les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui vont participer à l'amélioration de notre population. Cette méthode a l'avantage de permettre une convergence (plus) rapide des solutions, mais au détriment de la diversité des individus. On prend en effet le risque d'écarter des individus de piètre qualité, mais qui aurait pu apporter de quoi créer de très bonnes solutions dans les générations suivantes. Reste maintenant à définir comment nous allons les croiser.

Exemple : on a la population initiale suivant après l'application de la fonction de fitness :

Indvidu1	S1	S2	S3	S1
Individu 2	S1	S2	S3	S2
Individu 3	S1	S2	S3	S4
Individu 4	S1	S2	S3	S6
Individu 5	S3	S2	S3	S1
Individu 6	S3	S2	S3	S2
Individu 7	S5	S2	S3	S1
Individu 8	S5	S2	S3	S2
Individu 9	S6	S6	S7	S6
Individu 10	S7	S6	S5	S4

Tableau III.3: La population initiale

On suppose le taux de sélection est : 80%, alors le résultat :

Individu 1	S1	S2	S3	S1
Individu 2	S1	S2	S3	S2
Individu 3	S1	S2	S3	S4
Individu 4	S1	S2	S3	S6
Individu 5	S3	S2	S3	S1
Individu 6	S3	S4	S5	S6
Individu 7	S5	S2	S3	S1
Individu 8	S5	S2	S3	S2

Tableau III.4: La sélection

II.1.4.2 Le croisement

Le croisement, qui symbolise la reproduction sexuée (toujours par métaphore du mécanisme de sélection naturelle), est une des étapes importantes de l'A.G. dans notre approche proposé on a travaillé avec le croisement d'une seul point « single-point», par ce que c'est l'instrument majeur des innovations au sein de l'algorithme, c'est lui qui insuffle le changement. Il peut être effectué de plusieurs manières mais la plus courante croise les chaines de caractères de deux individus parents pour former des chaines de caractère enfants.

Exemple : On suppose le taux de croisement « une seul point »est : 50%, alors le résultat :

Individu 1	S1	S2	S3	S1
Individu 6	S3	S4	S5	S6

Après l'application de la croissement

Individu 1	S1	S2	S5	S6
Individu 6	S3	S4	S3	S1

Tableau III.5: Le croissement.

De façon aléatoire, un gène peut, au sein d'un chromosome être substitué à un autre. De la même manière que pour les enjambements, dans notre approche proposée et à la différence du croissement on définit le taux de mutation lors des changements de population qui est généralement compris entre 0.5% et 1% de la population totale.

Exemple : on suppose le taux de mutation est 5% alors le résultat :

Individu 1	S1	S2	S5	S6

Individu 1	S6	S2	S5	S1

Tableau III.6: la mutation

II.1.5 Existe-il un paramétrage universel ?

Non il n'existe pas de paramètres qui soient adaptés à la résolution de tous les problèmes qui peuvent être posés à une A.G. Cependant, certaines valeurs sont souvent utilisées peuvent être de bons points de départ pour démarrer une recherche de solution(s) à l'aide d'un A.G.

- La taille de la population initiale : entre 30 et 50 individus, Mais dans notre cas on a fixé en 20 individus.
- Le taux de croissement : entre 70% et 95%, Mais dans notre cas on a fixé en 70% comme paramètre par default avec la possibilité d’entrée via l’interface.
- Le taux de mutation : entre 0.5% et 1%, Mais dans notre cas on a fixé en 0.5% comme paramètre par default avec la possibilité d’entrée via l’interface.
- Pour la sélection on a fixé le taux de sélection on 80% comme paramètre par default avec la possibilité d’entrée via l’interface.
- Le nombre d’itération est 100 itérations comme paramètre par default avec la possibilité d’entrée via l’interface.

Conclusion

Les algorithmes génétiques fournissent des solutions proches de la solution optimale à l'aide des mécanismes de sélection, d'hybridation et de mutation. Ils sont applicables à de nombreux problèmes.
Dans le chapitre suivant nous avons détaillée l’implémentation de notre approche que nous avons proposée.

Chapitre IV

IMPLEMENTATION ET CONCEPTION

Introduction

D’après ce que n'a vu dans les chapitres précédant alors dans ce chapitre on va détaillée l’implémentation de notre approche.

I. Environnement d’expérimentation

L’environnement dans lequel nous avons réalisé notre application et obtenu les différentes expérimentations est défini par les éléments suivants :

I.1. Environnement matériel et logiciel : Nous avons développé notre application sur une machine fonctionnant sous le système Windows (Windows 8.1), avec les caractéristiques suivantes :

- Intel Core i5avec une vitesse de 2.5 GHz
- Capacité de mémoire 4 Go de RAM.

I.2 Environnement de développement : L’application est développée par le langage de programmation Java et on utilise le NetBeans IDE la version 8.0 avec pour gérer notre base de donnée on utilise oracle SQL*Plus.

- **Java :** Java est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable. Des ordinateurs portables aux centres de données, des consoles de jeux aux superordinateurs scientifiques, des téléphones portables à Internet, la technologie Java est présente sur tous les fronts.
- **NetBeans IDE :** NetBeans est un environnement de développement intégré (IDE) pour Java, placé en open source par Sun en juin 2000 sous licence CDDL (Common Développement and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, XML et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, éditeur graphique d'interfaces et de pages web).NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X et Open VMS. NetBeans est lui-même développé en Java, ce qui peut le rendre assez lent et gourmand en ressources mémoires.
- **Oracle SQL*Plus** est un utilitaire en ligne de commande d'Oracle qui permet aux utilisateurs d'exécuter interactivement des commandes SQL et PL/SQL. Décliné en plusieurs versions (graphique et web), il est principalement distribué avec le produit Oracle Database.

II. Notre base de données

Ces fichiers fournissent des données détaillées de la sécurité routière sur les circonstances de blessures accidents de la route en GB à partir de 1979, les types de véhicules impliqués et les pertes consécutifs. Les statistiques ne concernent que les accidents corporels sur les routes publiques qui sont signalés à la police, et ensuite enregistrés, en utilisant l'accident STATS19 formulaire de déclaration.

Elle comprend également : Résultats de données de dépistage souffle test de dispositifs de test de l'haleine numériques récemment introduites, comme prévu par les autorités de police en Angleterre et au Pays de Galles

Résultats du taux d'alcool dans le sang (milligrammes / 100 millilitres de sang) fournis par faire correspondre les données des coroners (fournies par les coroners en Angleterre et au Pays de Galles et par Procurators Fiscal en Ecosse) avec des données de mortalité à partir des données STATS19 de police des accidents de la route en Grande-Bretagne. Pour les cas où les niveaux d'alcoolémie pour une fatalité sont «inconnus» sont la conséquence d'une correspondance infructueuse entre les deux ensembles de données.

III. Les modèles proposer

III.1 Notre interface générale

Voici la figure suivante **Figure 22** (interface générale) qui il contient la table de placement, et estimer les informations de notre base et les requête que entré par l'utilisateur, tous sa fait juste après la connexion par la base de donnée. Et même le nombre des sites et des relations, la tempe d'exécution.

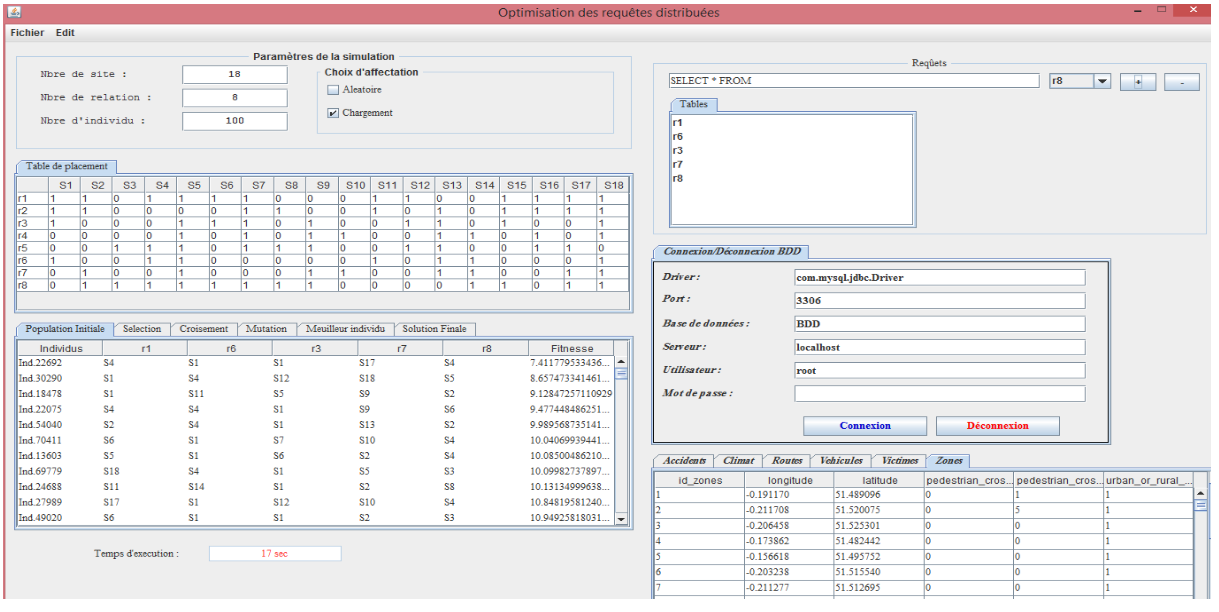


Figure IV.22: interface générale.

III.2 La fenêtre des paramètres de l'algorithme génétique

Dans cette fenêtre (Paramètres) nous présentons les différents paramètres de notre algorithme tell que le taux de sélection (Elitisme) et on préjuge une 80% comme valeur par défaut, taux de croisement on suppose une 7% également une valeur par défaut, taux de mutation on conjecture une 7% encore une valeur par défaut, le nombre des itérations égal à 10 itérations et le nombre des individus est 100 individus. Avec la possibilité de changer ces paramètres par l'utilisateur, et après on click dans le bouton valider le résultat c'est la fenêtre suivant.

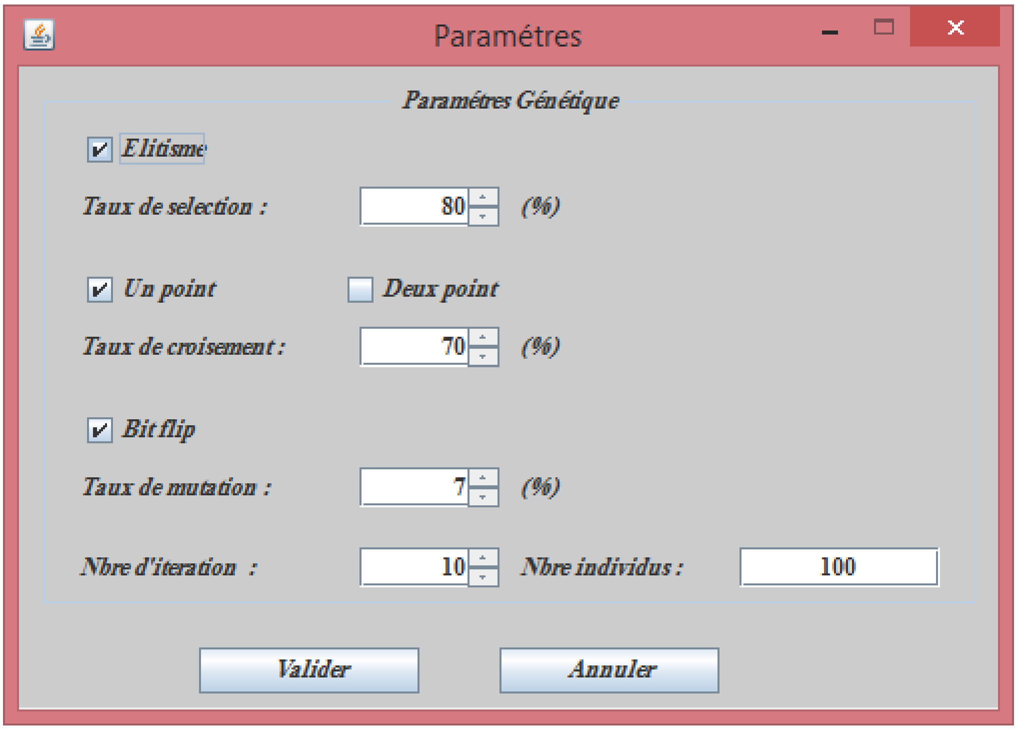
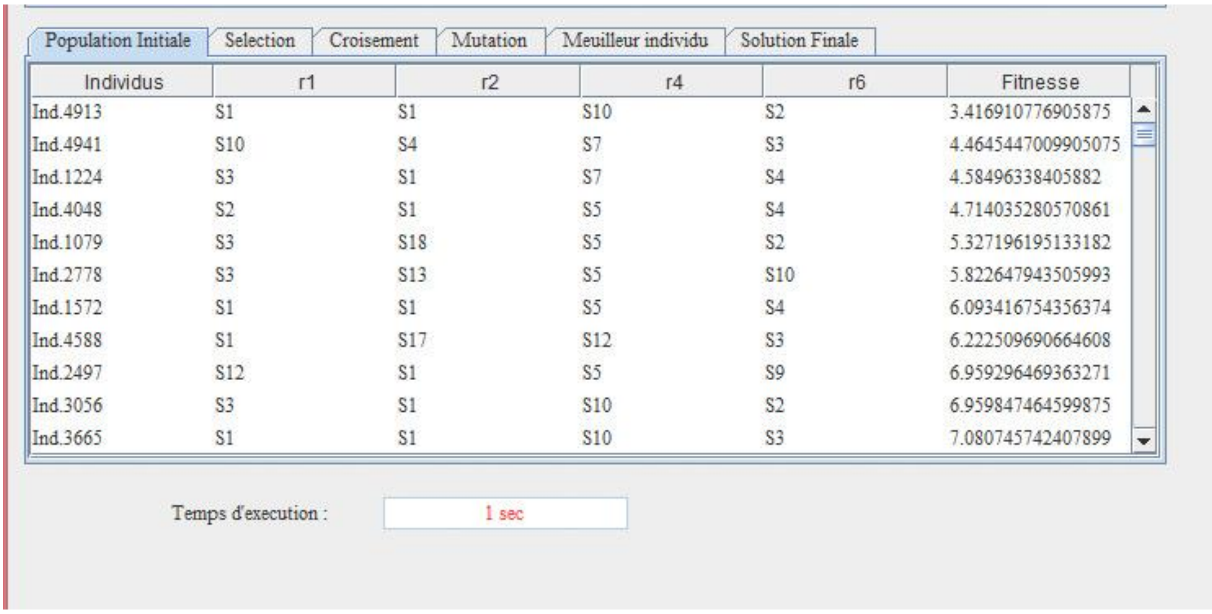


Figure IV.23: paramètre de l'algorithme génétique.

III.3 La population initiale

Après la fenêtre précèdent cette fenêtre c'est le résultat de la fenêtre précèdent et donne tous les cas (possibilité) des résultats pour la population initial avec leur fitness traie par ordre croissent.

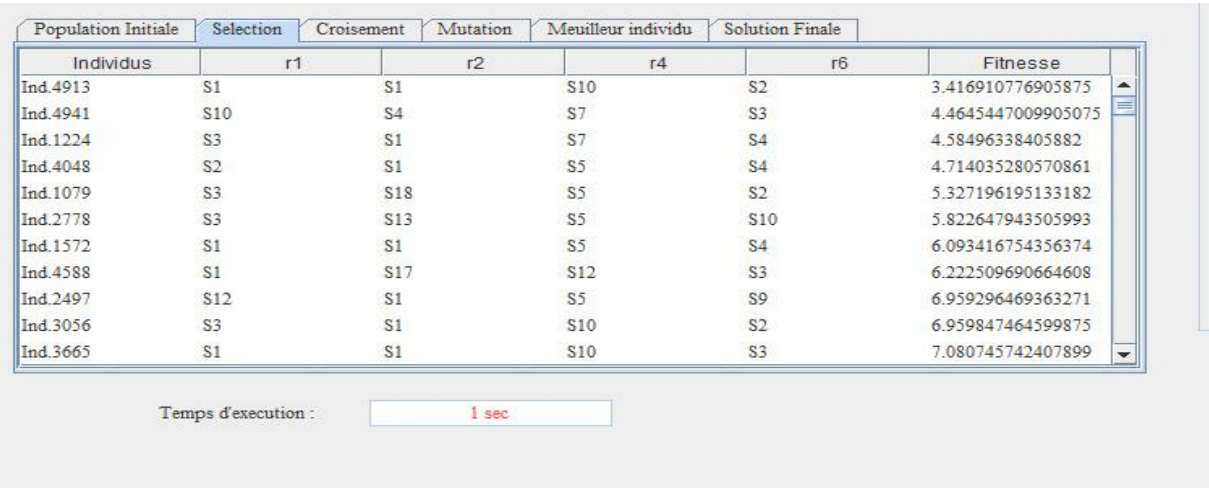


Individus	r1	r2	r4	r6	Fitnesse
Ind.4913	S1	S1	S10	S2	3.416910776905875
Ind.4941	S10	S4	S7	S3	4.4645447009905075
Ind.1224	S3	S1	S7	S4	4.58496338405882
Ind.4048	S2	S1	S5	S4	4.714035280570861
Ind.1079	S3	S18	S5	S2	5.327196195133182
Ind.2778	S3	S13	S5	S10	5.822647943505993
Ind.1572	S1	S1	S5	S4	6.093416754356374
Ind.4588	S1	S17	S12	S3	6.222509690664608
Ind.2497	S12	S1	S5	S9	6.959296469363271
Ind.3056	S3	S1	S10	S2	6.959847464599875
Ind.3665	S1	S1	S10	S3	7.080745742407899

Figure IV.24: la population initiale.

III.4 La sélection

Après la validation de la paramètres de notre algorithme pour obteniez la solution optimale « de notre exemple proposé bien sûr », le résultat de la sélection dans la figure suivant :

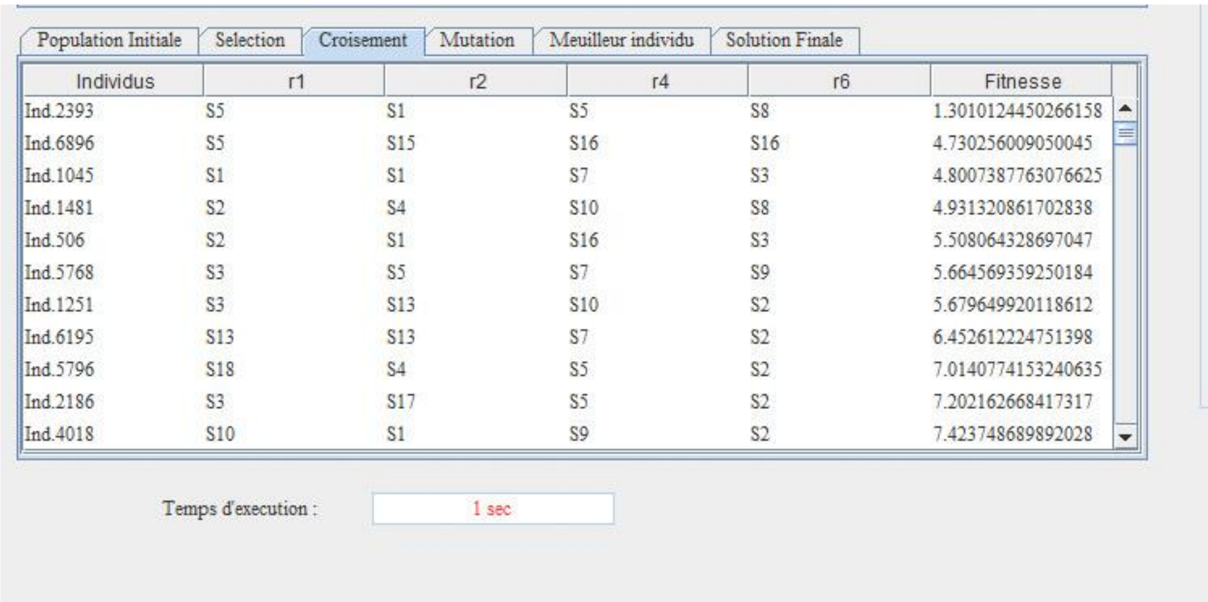


Individus	r1	r2	r4	r6	Fitnesse
Ind.4913	S1	S1	S10	S2	3.416910776905875
Ind.4941	S10	S4	S7	S3	4.4645447009905075
Ind.1224	S3	S1	S7	S4	4.58496338405882
Ind.4048	S2	S1	S5	S4	4.714035280570861
Ind.1079	S3	S18	S5	S2	5.327196195133182
Ind.2778	S3	S13	S5	S10	5.822647943505993
Ind.1572	S1	S1	S5	S4	6.093416754356374
Ind.4588	S1	S17	S12	S3	6.222509690664608
Ind.2497	S12	S1	S5	S9	6.959296469363271
Ind.3056	S3	S1	S10	S2	6.959847464599875
Ind.3665	S1	S1	S10	S3	7.080745742407899

Figure IV.25: la sélection.

III.4 Le croissement

Après l'étape de la sélection on va prend les meilleurs 70% permet les individus pour faire le croissement et finalement on a obtenu la figure suivant.

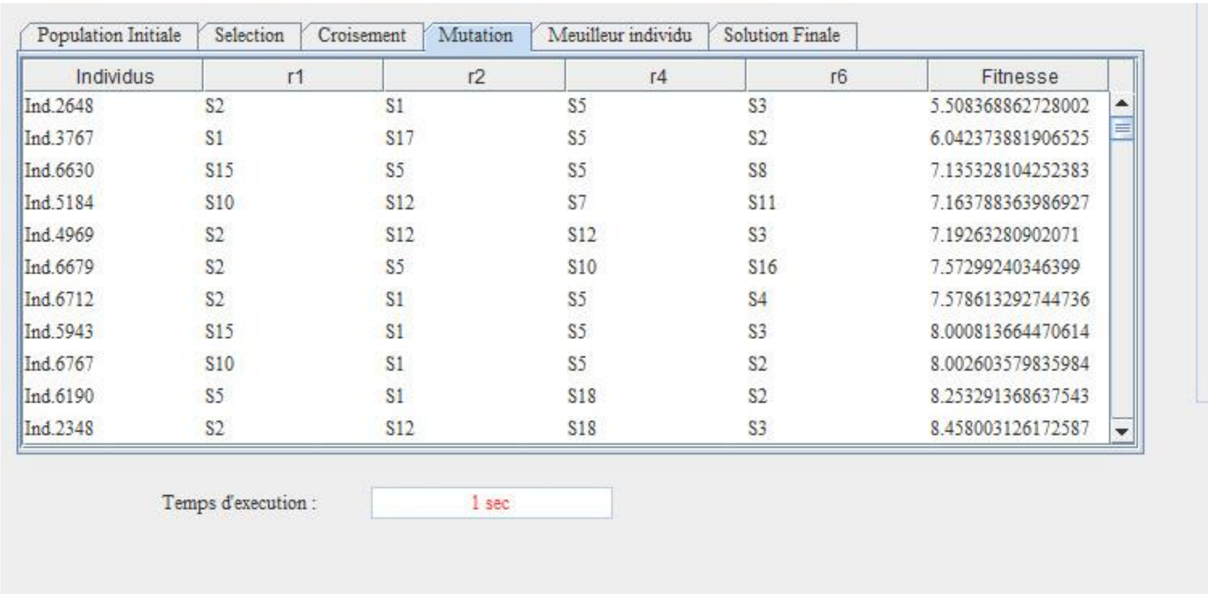


Individus	r1	r2	r4	r6	Fitnesse
Ind.2393	S5	S1	S5	S8	1.3010124450266158
Ind.6896	S5	S15	S16	S16	4.730256009050045
Ind.1045	S1	S1	S7	S3	4.8007387763076625
Ind.1481	S2	S4	S10	S8	4.931320861702838
Ind.506	S2	S1	S16	S3	5.508064328697047
Ind.5768	S3	S5	S7	S9	5.664569359250184
Ind.1251	S3	S13	S10	S2	5.679649920118612
Ind.6195	S13	S13	S7	S2	6.452612224751398
Ind.5796	S18	S4	S5	S2	7.0140774153240635
Ind.2186	S3	S17	S5	S2	7.202162668417317
Ind.4018	S10	S1	S9	S2	7.423748689892028

Figure IV.26: croissement.

III.5 La mutation

Après l'étape de croisement an a obtenu les résultats pour faire la mutation entre les individus aléatoirement et voici les résultats dans la figure suivant.

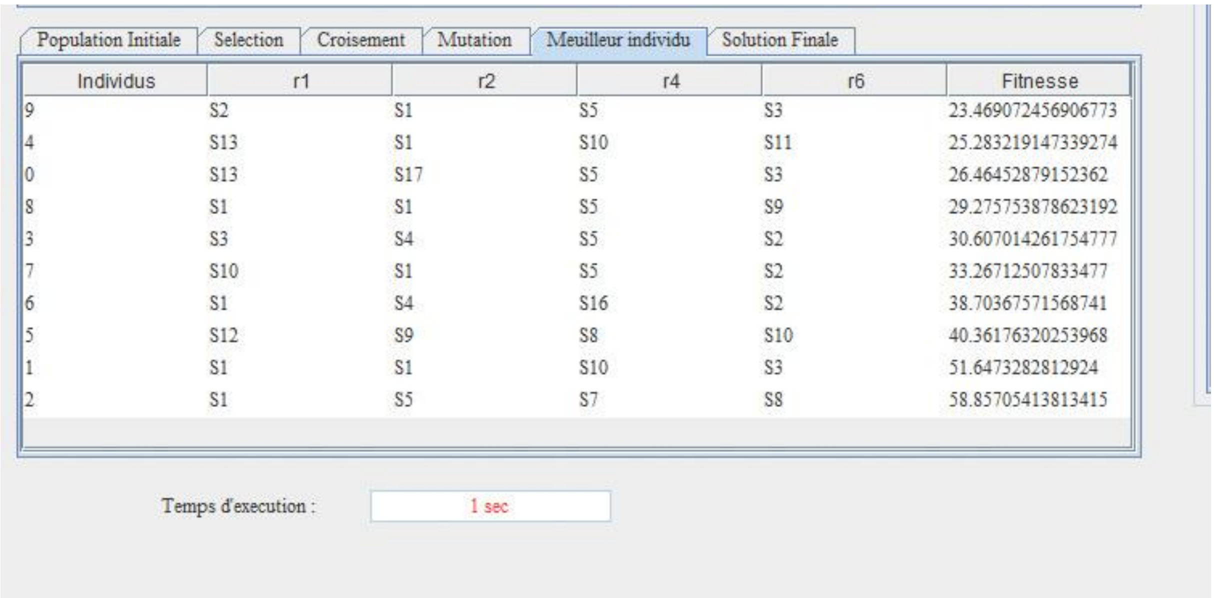


Individus	r1	r2	r4	r6	Fitnesse
Ind.2648	S2	S1	S5	S3	5.508368862728002
Ind.3767	S1	S17	S5	S2	6.042373881906525
Ind.6630	S15	S5	S5	S8	7.135328104252383
Ind.5184	S10	S12	S7	S11	7.163788363986927
Ind.4969	S2	S12	S12	S3	7.19263280902071
Ind.6679	S2	S5	S10	S16	7.57299240346399
Ind.6712	S2	S1	S5	S4	7.578613292744736
Ind.5943	S15	S1	S5	S3	8.000813664470614
Ind.6767	S10	S1	S5	S2	8.002603579835984
Ind.6190	S5	S1	S18	S2	8.253291368637543
Ind.2348	S2	S12	S18	S3	8.458003126172587

Figure IV.27: la mutation.

III.6 Meilleur individus

Après on a découvrir les meilleurs individus mentionnés ci-dessous.

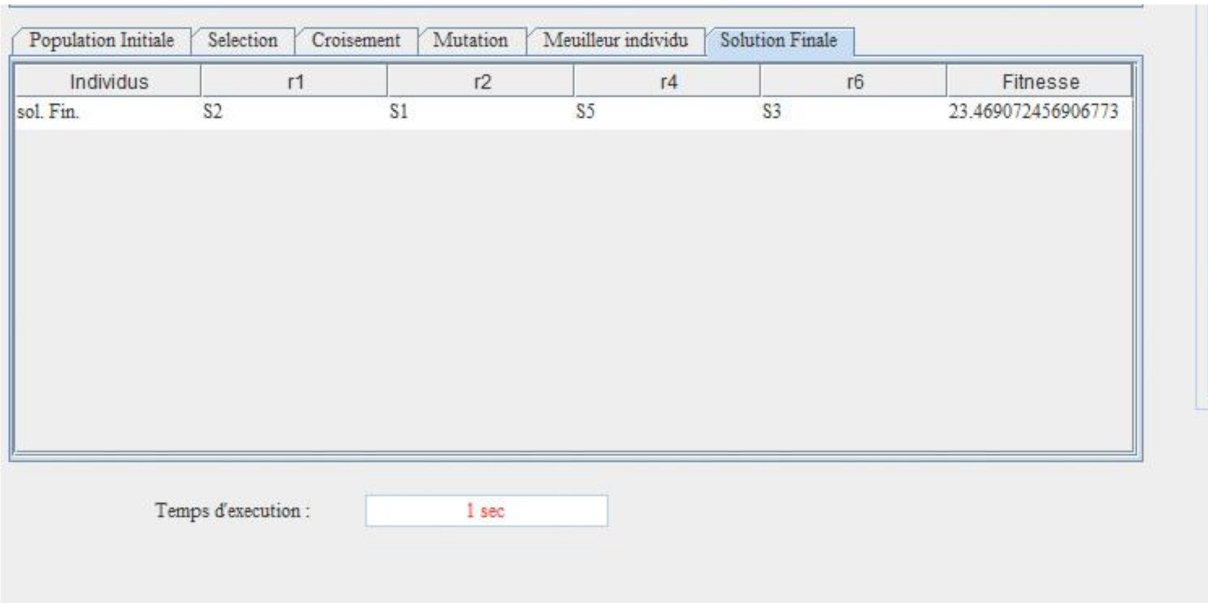


Individus	r1	r2	r4	r6	Fitnesse
9	S2	S1	S5	S3	23.469072456906773
4	S13	S1	S10	S11	25.283219147339274
0	S13	S17	S5	S3	26.46452879152362
8	S1	S1	S5	S9	29.275753878623192
3	S3	S4	S5	S2	30.607014261754777
7	S10	S1	S5	S2	33.26712507833477
6	S1	S4	S16	S2	38.70367571568741
5	S12	S9	S8	S10	40.36176320253968
1	S1	S1	S10	S3	51.6473282812924
2	S1	S5	S7	S8	58.85705413813415

Figure IV.28: meilleur individus.

III.7 La solution finale

On a obtenu la solution finale après le trie bien sûr par ordre ascendant.



Individus	r1	r2	r4	r6	Fitnesse
sol. Fin.	S2	S1	S5	S3	23.469072456906773

Figure IV.29: la solution finale.

III.8 Graphe de notre exemple

Après les étapes précédentes la figure suivant donne les résultats par diagramme à ligne brisée.

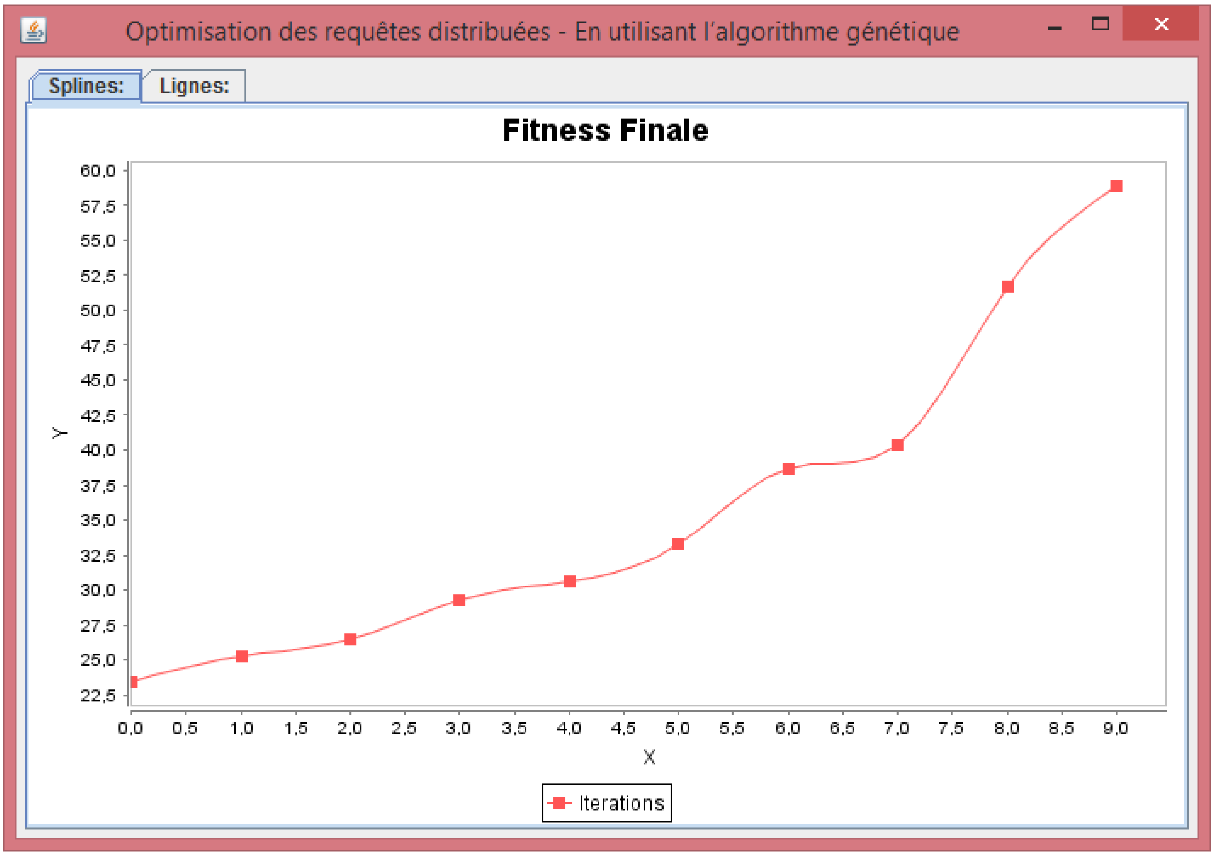


Figure IV.30: graphe de notre exemple.

Conclusion

Dans ce chapitre nous avons fait plusieurs expérimentations pour évaluer notre approche. Nous avons pu voir l'influence des différents paramètres proposés sur le comportement de l'approche. D'après les résultats obtenus, nous avons pu constater que les paramètres utilisés dans notre approche ont donné une vision générale sur le comportement de l'approche.

CONCLUSION GENERALE

CONCLUSION GENERALE

Dans ce rapport, nous avons étudié le problème d'optimisation de requête distribuée et les caractéristiques de l'optimiseur, nous avons vu que la complexité d'optimisation augmente lorsque le nombre de relations et sites augmente dans une requête, alors la recherche d'une stratégie optimale pour exécuter une requête à mener à développer des stratégies d'optimisation qui constituent la partie centrale de l'optimiseur, ces stratégies peuvent être vu en tant que trois classes. Nous avons présentés les stratégies déterministes qui ont le temps exponentiels et la complexité de l'espace, les stratégies randomisées qui évitent le coût élevé d'optimisation en termes de consommation mémoire et de temps et comme on a les algorithmes génétiques ont l'énorme avantage de pouvoir être appliqués dans un grand nombre de domaines de recherche de solution, lorsqu'il n'est pas nécessaire d'avoir la solution optimale, qui prendrait par exemple trop de temps et de ressources pour être calculée

Finalement les stratégies génétiques connues par leurs capacités de traiter différents problèmes d'optimisation combinatoires mais basé sur les paramètres aléatoire.

REFERENCES

Bibliographies:

[1]: Alaa Aljanaby, Emad Abuelrub and Mohammed Odeh "A Survey of Distributed Query Optimization", The international Arab journal of Information Technology, Vol.2, No.1, January **(2005).**

[2]: A. Hameurlain, F. Morvan "Evolution of Query Optimization Methods", Trans. on Large Scale Data and Knowledge Cent. Syst.I LNCS 5740, pp211-242, **(2009).**

[3]: A. Hameurlain, « Evolution of Query Optimization Methods: From Centralized Database Systems to Data Grid Systems », Proceedings of the 20th International Conference on Database and Expert Systems Applications, **(2009)**.

[4]: Chihping Wang, Ming-Syan Chen, "On Complexity of Distributed Query Optimization", IEEE Transaction on knowledge data engineering, Vol 8 N° 4, August **(1996).**

[5]: Franck Morvan, Abdelkader Hameurlain "Apport des agent's mobiles à l'optimisation de requêtes réparties à grande echelle", Institut de Recherche en Informatique de Toulouse.

[6]: Kristin Bennett, Michael C. Ferriset Yannis E. Ioannidis, « A Genetic Algorithm for Database Query Optimization », Computer Sciences Department University of Wisconsin 1210 West Dayton Street Madison, Wisconsin 53706.

[7]: L.M.Hass, "R*: A research project on distributed relational DBMS",Database Engineering, Vol.5, **(1982).**

[8]: Majid Khan and M. N. A. Khan "Exploring Query Optimization Techniques in Relational Databases" SZABIST, Islamabad, Pakistan, **(2013)**.

[9]: Michael Steinbrunn and Guido Moerkotte and Alfons Kemper "Heuristic and randomized optimization "The VLDB Journal vol .6, pp.191–208, **(1997)**.

[10]: OSZU, M.T and Valduriez P "Principles of Distributed database systems", livre 2eme edition, Prentice Hall International, NJ, **(1999)**.

[11] : P.A. Bernstein, N. Goodman, E.Wona, C. Reeve, and T.B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-l)," ACM Trans. Data base Système, vol. 6, no.4, pp. 602-625, Décembre **(1981)**.

[12]: Pankti Doshi and Vijay Raisinghani "Review of Dynamic Query optimization Strategies in Distributed Database" Electronics Computer Technology (ICECT), 3rd International Conference on Kanya kumariVol 6, pp145-149, April **(2011)**.

[13] : **Pawandeep Kaur, Jaspreet Kaur Sahiwal,** « Join Query Optimization in Distributed Databases »,International journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013 ISSN 2250-3153. **(2013).**

[14]: Reza Ghaemi Amin Milani Fard, Hamid Tabatabaeeand Mahdi Sadeghizadeh, "Evolutionary Query Optimization for Heterogeneous Distributed Database Systems", World Academy of science, Engineering and Technology. Vol: 2, (**2008**).

[15]: Yannis E. Ioannidis and YounkyungCha Kang " Randomized algorithms for optimizing large join queries", in Proceedings of the ACM SIGMOD Conference on Management of Data, Atlantic City, USA, pp 312-321, **(1990).**

[16]: X.Lin, "Query Optimization Strategies and Implementation Based on Distributed Database», Computer Science and Information Technology, 2nd IEEE International conference, **(2009).**

WEBGRAPHE

[17] : Site web : « Sciences des données : De la logique du premier ordre à la Toile », lien : http://books.google.fr/books?id=LxhFxroanSQC&pg=PT22&dq=%22Optimisation+de+requ%C3%AAte%22&hl=fr&sa=X&ei=PN48Uc4UspLsBoWgWg&ved=0CDUQ6AEwAQ.Consulter le 26/01/2015

[18] : Site web : « Les bases de données distribuées : Qu'est-ce que c'est ? », lien : http://substance.etsmtl.ca/les-bases-de-donnees-distribuees-une-evolution-qui-pose-des-problemes/. Consulter le 26/01/2015.

[19] : Site web : « Bases de données distribuées et fédérées », Consulter le 27/01/2015

[20] : Site web : memoireonline.com : « Bases de données reparties sous Oracle », Ecole supérieur de management commerce et informatique, Maroc - Ingénierie en informatique 2008 lien : http://www.memoireonline.com/05/10/3459/m_Bases-de-donnees-reparties-sous-Oracle1.html consulter le 23/01/2015