الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـلـيـــــة الـعـلـــــوم
FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

Mémoire de MASTER

Domain : Mathématiques et Informatique
Filière : Informatiques
Option : Systèmes d'Information et de Décision

Par:
Fantazi Elkhansa Et Rouadi Nabila

THEME

APPLICATION DES ALGORITHMES GENETIQUES POUR L'OPTIMISATION DES REQUÊTES

Soutenu publiquement le 11-06-2017 devant le jury composé de:

M_{me} Fatna GUIBADJ	*M.C.(A)*	*Président*
Mr Noureddine CHAIB	*M.C.(A)*	*Examinateur*
Mr Laradj CHELLAMA	*M.C.(A)*	*Encadreur*

Année Universitaire 2016/2017

Remerciements

*Toute notre gratitude et remerciements au bon **DIEU** qui nous a donné la force, le courage et la volonté d'élaborer ce travail.*

Nous adressons notre profond remerciement à Monsieur

*L'encadreur **CHELLAMA LARADJ** pour ses précieux conseils et pour avoir soutenus, dirigés et orientés durant toute la période de notre projet.*

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail Et de l'enrichir par leurs évaluations.

Nous remercions tous nos enseignant de département d'Informatique.

Remerciement les plus sincères à toutes les personnes qui nous ont aidé de près ou de loin à accomplir notre travail.

Dédicace

Je dédie ce mémoire

A mes chers parents Pour leur patience, leur amour, leur soutien et leurs Encouragements,

A mes chers frères et sœurs,

A toute ma chère famille,

A mes professeurs,

A mes chers amis,

A tous ceux qui m'aiment,

A tous ceux que j'aime,

A tous ceux qui m'ont aidé de prés ou de loin,

Je dédie ce travail avec hommage.

Rouadi Nabila

Dédicace

Louange à Dieu, le seul et l'unique

À mes très chers parents, qui ont fait de moi ce que je suis

À mes chers frères et sœurs.

À tous mes amies

À tous mes familles FANTAZI et TERBAH

À tous mes amis en témoignage de l'amitié sincère qui nous a

Liées et des Bons moments passés ensemble

À tous ceux qui me sont chers

À tous ceux qui aiment EL KHANSA et ceux qui EL KHANSA aime.

Fantazi ELkhansa

Résumé

L'optimisation des requêtes reçoit une attention considérable de la part des chercheurs. L'utilisation des bases de données relationnelles dans plusieurs applications entraine, naturellement, une croissance au niveau des attentes et des exigences en termes d'efficience.

L'objectif de cette étude vise une contribution dans le domaine de l'optimisation des requêtes SQL en utilisant les algorithmes génétique pour permettre une convergence rapide vers des plans d'exécution de requêtes améliorés et il est indispensable au traitement des requêtes dans un temps raisonnable.

Notre travail consiste à l'application des algorithmes génétiques pour générer tous les plans d'exécution possibles et d'en sélectionner le meilleur plan en terme de couts.

Nous évaluons notre approche de recherche de plan optimum, à travers d'expérimentations réalisées à l'aide du langage Java combiné avec le SGBD SQL server pour l'implémentation de notre requête d'essai.

Mots clés : Java, SQL server, Algorithme génétique, plans d'exécutions.

Abstract

The optimization of requests receives considerable attention from researchers. The use of relational databases in several applications naturally leads to growth in expectations and requirements in terms of efficiency.

The objective of this study is to contribute to the optimization of SQL queries by using genetic algorithms to allow rapid convergence to improved query execution plans and is essential for the processing of queries within a reasonable time.

Our work consists in applying genetic algorithms to generate all the possible execution plans and to select the best plan in terms of costs.

We evaluate our optimum plan search approach through experiments carried out using the Java language combined with the SQL server DBMS for the implementation of our test request.

Keywords: Java, SQL server, Genetic algorithm, execution plans

ملخص

حظي الاستعلام الأمثل باهتمام كبير من جانب الباحثين. إن استخدام قواعد البيانات العلائقية في العديد من التطبيقات يؤدي الى نمو التوقعات والمتطلبات بشكل طبيعي من حيث الكفاءة.

الهدف من هذه الدراسة هو المساهمة في مجال تحسين استعلامات الأسكيال باستخدام الخوارزميات الجينية من اجل تسريع تنفيذ الخطط وتحسين الاستعلامات لأنه من الضروري علاج الاستعلامات في غضون فترة زمنية معقولة.

يتمحور عملنا في تطبيق الخوارزميات الجينية لتوليد خطط تنفيذية ممكنة وإختيار أفضل خطة من حيث التكاليف.

تم تقيم بحث نهجنا للخطة المثالية من خلال تجارب أجريت باستخدام جافا مع قواعد بيانات الخادم اسكيال سيرفر لتنفيذ مثال التطبيق.

الكلمات المفتاحية: الخادم أسكيال, جافا, الخوارزميات الجينية, خطط التنفيذ.

Table des Matières

Table des Matières vii

Liste de figures x

1 Optimisation de requêtes 4

1 Introduction 5

2 Processus d'optimisation de requêtes 6

3 Les phases d'optimisation: 9

3.1 Optimisation logique ou Réécriture 9

3.1.1 Règles de transformation des arbres 11

3.2 Optimisation physique 14

3.2.1 Boucles imbriquées 15

3.2.2 Tri-fusion 15

3.2.3 Hachage 17

4 La Recherche du meilleur plan 18

5 Stratégies de recherche 19

5.1 Programmation dynamique (DP) 20

5.2 Recuit simulé 21

5.3 Algorithme Génétique 21

6 Conclusion 22

2 Les algorithmes génétiques 23

1 Introduction 24
2 Les Concepts importants des algorithmes génétiques 25
3 Fonctionnement d'un algorithme génétique 25
4 Caractéristiques des algorithmes génétiques 26
4.1 Codage des données 26
4.2 La fonction d'évaluation 28
5 Les opérateurs génétiques 28
5.1 L'opérateur de sélection 28
5.1.1 La loterie biaisée ou roulette wheel 29
5.1.2 Méthode élitiste 30
5.1.3 La sélection par tournois 30
5.1.4 La sélection universelle stochastique 30
5.2 L'opérateur de croisement ou crossover 31
5.3 L'opérateur de mutation 32
6 Conclusion 33

3 Présentation et implémentation de notre approche 34
1 Introduction 35
2 Les outils utilisés 35
3 Description de l'AG d'optimisation de requête 38
3.1 L'Algorithme génétique d'optimisation de requête 38
3.2 La génération de population initiale 39
3.3 La fonction d'adaptation(Fitness) 39
3.4 Les opérateurs génétiques 40
3.4.1 Sélection: 40
3.4.2 Croisement: 40
3.4.3 Mutation : 40
4 Implémentation de notre approche 41
5 Conclusion 46

Conclusion générale et perspective 47

Bibliographie 48

Liste de figures

1.1 Processus d'optimisation de requête basée sur le coût [1] 7
1.2 Requête SQL [2] 10
1.3 L'Arbre Algébrique Logique [2] 11
1.4 Commutativité des jointures [3] 12
1.5 Associativité des jointures [3] 12
1.6 Fusion des projections [3] 12
1.7 Regroupement des restrictions [3] 12
1.8 Quasi-commutativité des restrictions et des projections [3] . . . 13
1.9 Quasi-commutativité des restrictions et des jointures [3] 13
1.10 Commutativité des restrictions avec les unions, intersections ou différences [3] 13
1.11 Quasi-commutativité des projections et jointures (la projection ne doit pas perdre les attributs de jointure) [3] 14
1.12 Commutativité des projections avec les unions [3] 14
1.13 Algorithme des boucles imbriquées [3] 15
1.14 Algorithme de tri-fusion: phase de tri [4] 16
1.15 6 Algorithme de tri-fusion: la phase de fusion [4] 16
1.16 Algorithme par hachage [3] 17
1.17 Processus général d'optimisation [4] 19

2.1 Fonctionnement d'un Algorithme génétique 26

2.2 Schéma d'une roulette [5] 29
2.3 Croisement en un point d'un individu de 14 bits [5] 31
2.4 Croisement en deux point d'un individu de 14 bits [5] 31
2.5 Croisement uniforme [5] 32
2.6 Mutation classique [6] 32
2.7 Mutation adaptative [6] 33

3.1 Les plan d'exécutions équivalents 38
3.2 Estimation de coût d'exécution 41
3.3 1er itération 43
3.4 Dernier itération 44

Introduction Générale

Introduction Générale

Les problèmes d'optimisation occupent actuellement une place importante dans la communauté scientifique, un problème est défini par un ensemble de variables, une fonction objectif (fonction de coût) et un ensemble de contraintes.

Dans le domaine des bases de données relationnelles, un SGBD analyse, optimise et exécute une requête utilisateur dans laquelle on n'indique ni les algorithmes à appliquer, ni les chemins d'accès aux données, le système a toute latitude pour déterminer ces derniers et les combiner de manière à obtenir les meilleures performances.

Dans cette optique, plusieurs approches et métaheuristiques ont été proposés dans la littérature pour alléger l'optimiseur d'un SGBD afin d'obtenir les meilleurs performances possibles en terme de temps de réponse.

Le plan du présent manuscrit est le suivant :

Dans le premier chapitre, nous allons définir les notions de l'optimisation d'une requêtes relationnelle, puis nous allons exposer les étapes nécessaire pour la tâche d'un optimiseur afin de sélection un plan d'exécution optimal à partir d'un espace de recherche.

Dans le deuxième chapitre, nous présentons l'algorithme génétique comme métaheuristiques pour l'optimisation d'un problème difficile, en décrivant le principe de base des ces différents opérateurs (selection, croisement,mutation,...) .

Le troisième chapitre présente notre démarche d'application de l'algorithme génétique pour générer l'espace de recherche à partir d'une population initiale contenant quatre (04) plans aléatoire d'une requête de jointure de sept tables (07), aussi nous exposons les étapes à suivre pour avoir le meilleur plan d'exécution. Enfin, les résultats de notre expérimentation s'avèrent prometteuses.
Nous concluons, et présentons les perspectives de ce travail dans le dernier chapitre.

Chapitre 1

Optimisation de requêtes

1 Introduction

L'optimisation des requêtes a été un domaine de recherche actif depuis le début du développement de DBMSs relationnelle (Database Management System).

Depuis les années 1975, l'optimisation de requêtes dans les bases de données relationnelles a reçu une attention considérable[7] [8]. Les systèmes ont beaucoup progressé. Alors que les premiers étaient très lents, capables seulement d'exécuter quelques requêtes par seconde sur les bases de données du benchmark TPC/A ou B, ils supportent aujourd'hui des milliers de transactions par seconde. Bien sûr, cela n'est pas dû seulement à l'optimiseur, mais aussi aux progrès du matériel (les temps d'entrée-sortie disque restent cependant de l'ordre de la dizaine de millisecondes) et des méthodes d'accès. L'optimiseur est donc un des composants essentiels du SGBD relationnel ; avec les nouveaux SGBD objet relationnel ou objet, il devient encore plus complexe.

Classiquement, un optimiseur transforme une requête exprimée dans un langage source (SQL) en un plan d'exécution composé d'une séquence d'opérations de bas niveau réalisant efficacement l'accès aux données.
Le langage cible est donc constitué d'opérateurs de bas niveau, souvent dérivés de l'algèbre relationnelle complétée par des informations de niveau physique, permettant de déterminer l'algorithme choisi pour exécuter les opérateurs [9] . Ces informations associées à chaque opérateur, appelées annotations, dépendent bien sûr du schéma interne de la base (par exemple, de l'existence d'un index) et de la taille des tables. Elles complètent utilement l'algèbre pour choisir les meilleurs chemins d'accès aux données recherchées. Au-delà de l'analyse de la requête , l'optimisation de requêtes est souvent divisée en deux phases :
l'optimisation logique, qui permet de réécrire la requête sous une forme

canonique simplifiée et logiquement optimisée, c'est-à-dire sans prendre en compte les coûts d'accès aux données, et l'optimisation physique, qui effectue le choix des meilleurs algorithmes pour les opérateurs de bas niveau compte tenu de la taille des données et des chemins d'accès disponibles.

L'optimisation logique reste au niveau de l'algèbre relationnelle, alors que l'optimisation physique permet en particulier d'élaborer les annotations.

Les optimisations logique et physique ne sont malheureusement pas indépendantes ; les deux peuvent en effet changer l'ordre des opérateurs dans le plan d'exécution et modifier le coût du meilleur plan retenu.

2 Processus d'optimisation de requêtes

L'optimiseur a deux tâches principales à accomplir en un temps acceptable: d'abord, il faut trouver les solutions alternatives pour une même requête donnée. Ensuite, parmi ces alternatives, il faut choisir, en se basant sur certains critères, la plus performante pour exécuter la requête. Pour mieux comprendre les problématiques existantes dans ce contexte, nous devons nous rappeler le processus général pour traiter une requête.

Une requête est une expression décrivant les informations que l'utilisateur recherche parmi les sources de données. Elle ne décrit que synthétiquement les informations recherchées, sans préciser la façon d'y accéder.
Le système doit donc traduire l'expression automatiquement, via son module d'analyse de requêtes, en une séquence d'opérations élémentaires, appelée plan d'exécution, dont l'exécution pourra retourner les résultats envisagés. Mais la traduction n'est pas unique. En effet, plusieurs plans d'exécution peuvent être équivalents, c'est à dire donner le même résultat pour la requête. L'ensemble de

ces traductions (plans d'exécution) forme un espace de recherche. L'optimiseur doit employer un modèle de coût pour prévoir le coût d'exécution de chaque plan d'exécution afin d'en choisir celui avec un coût minimal. Néanmoins, l'exploration de l'espace de recherche est parfois difficile pour les requêtes complexes car il existe tellement de solutions possibles que l'optimiseur doit utiliser une stratégie de recherche pour trouver la solution idéale en un temps acceptable, en évitant d'explorer tout l'espace de recherche exhaustivement.

La figure 3.2 présente un Processus d'optimisation de requête basée sur le coût.

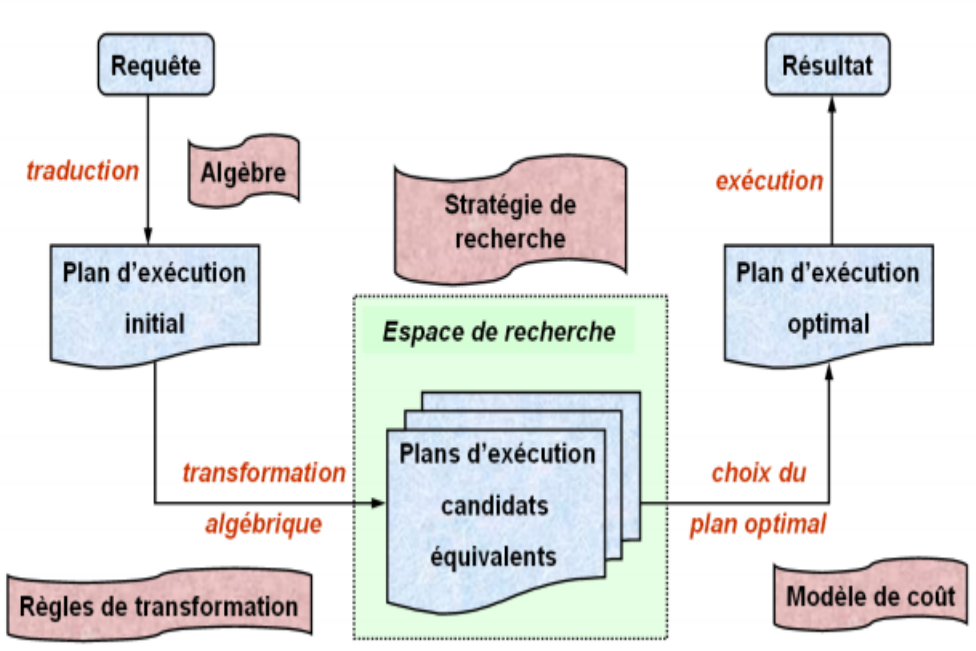


Figure 1.1: Processus d'optimisation de requête basée sur le coût [1]

Algèbre : Une algèbre utilisée par l'optimiseur est constituée d'opérateurs relationnels simples (ex. sélection, projection, jointure, union, etc.) et d'un opérateur de communication entre les sources et les autres composants du système. Les opérateurs de l'algèbre acceptent en opérande des relations composées de tuples, et produisent des relations en tant que résultat de l'opérateur. Dans les systèmes relationnels, l'algèbre a été définie par Codd [10] . Chaque requête est traduite dans les opérateurs de l'algèbre. Il existe d'autres algèbres pour les systèmes non-relationnels, par exemple, une algèbre LORE a été proposée pour XML dans [11] .

Plan d'exécution : La requête peut être présentée en utilisant l'algèbre de l'optimiseur, par un arbre d'opérations dont les nœuds sont des opérateurs de l'algèbre, et les feuilles représentent les données des sources (opérandes). Cet arbre est appelé le plan d'exécution logique de la requête parce qu'il spécifie la méthode pour exécuter la requête, en indiquant l'ordonnancement de l'exécution des différents opérateurs, et la communication des données entre les sources et le système qui traite la requête. Si nous ajoutons les informations concernant l'exécution des opérateurs sur la couche physique du système, par exemple, l'algorithme utilisé pour effectuer les opérateurs binaires, ce plan d'exécution devient le plan d'exécution physique, qui est prêt à être exécuté par le système.

Règle de transformation : Les équivalences supportées par les opérateurs induisent des réécritures algébriques conservant la sémantique de la requête mais potentiellement plus simples à évaluer. Ces équivalences sont décrites par des règles de transformations. Certaines règles ont pour but de reformuler directement la requête en un plan d'exécution. D'autres utilisent un plan d'exécution initial pour générer un ensemble de plans équivalents.

Espace de recherche : L'ensemble des plans d'exécution équivalents constitue l'ensemble des possibilités d'exécution d'une requête. Le nombre de possibilités peut s'avérer rédhibitoire. Pour résoudre ce problème, l'optimiseur a besoin d'une stratégie de recherche.

Modèle de coût : Le but d'un modèle de coût est d'estimer le coût d'exécution des plans. Il permet ainsi de choisir le meilleur plan d'exécution ayant le moindre coût d'exécution. Le modèle de coût contient, d'une part, des statistiques sur les données et sur le système SGBD et, d'autre part, des formules pour évaluer la taille des résultats intermédiaires et le coût des plans. Ces formules reposent en général sur un certain nombre de paramètres et d'hypothèses simplificatrices. L'unité de mesure du coût dépend de l'objectif d'optimisation.

Stratégie de recherche.: La stratégie de recherche explore l'espace de recherche pour trouver le meilleur plan d'exécution en utilisant le modèle de coût. Cette stratégie définit quels sont les plans explorés parmi l'ensemble des plans de l'espace de recherche, et l'ordre dans lequel ces plans sont explorés. il existe plusieurs stratégies comme la Programmation dynamique , Recuit simulé, la stratégie déterministe, Algorithme génétique ces stratégies le plus utilisée pour construire les plans qui sont des arbres de jointures .

3 Les phases d'optimisation:

3.1 Optimisation logique ou Réécriture

L'optimisation consiste à réorganiser l'arbre en utilisant les propriétés algébriques de façon à produire un arbre équivalent (c'est à dire produisant la même réponse) mais dont l'évaluation sera plus efficace.

L'Arbre Algébrique Logique définit l'organisation structurée des opérateurs, telle que :

- Pour chaque nœud (sauf les feuilles) de l'arbre, il a une entrée représentant les données que l'opérateur manipule et une sortie représentant le résultat de l'exécution de l'opérateur.
- La racine représente l'opérateur fournissant le résultat final de la requête.
- Les feuilles représentent les données de source accédées par la requête.
- L'ordre de l'exécution des opérateurs est défini par l'arbre : depuis les feuilles vers la racine.

L'idée de base est de réduire au maximum la taille des relations intermédiaires produites. Pour cela, il faut utiliser le plus possible les sélections et les projections et les faire le plus tôt possible. Il faut donc faire "descendre" les sélections et projections dans l'arbre et rajouter éventuellement des projections permettant de ne manipuler que les attributs effectivement utilisés dans l'expression. La Figure 1.3 illustre l'Arbre Algébrique Logique correspondant à la requête citée dans la Figure 1.2.

```
select titre
from   Film f, Role r
where  nom_role ='John Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

Figure 1.2: Requête SQL [2]

Cette requête correspond aux opérations suivantes: une jointure entre les rôles et les films, une sélection sur les films (année=1958), une sélection sur les rôles ('John Ferguson), enfin une projection pour éliminer les colonnes non désirées.

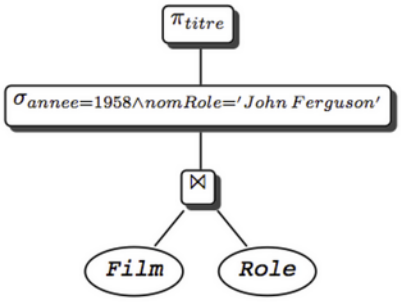


Figure 1.3: L'Arbre Algébrique Logique [2]

3.1.1 Règles de transformation des arbres

Le plan d'exécution logique (PEL) présentant, sous une forme normalisée (par exemple, les projections, puis les sélections, puis les jointures) les opérations nécessaires à l'exécution d'une requête donnée.
On peut reformuler le PEL grâce à l'existence de propriétés sur les expressions de l'algèbre relationnelle.

Ces propriétés appelées lois algébriques ou encore règles de réécriture permettent de transformer l'expression algébrique en une expression équivalente et donc de réagencer l'arbre. Le PEL obtenu est équivalent, c'est-à-dire qu'il conduit au même résultat. En transformant les PEL grâce à ces règles, on peut ainsi obtenir des plans d'exécution alternatifs, et tenter d'évaluer lequel est le meilleur. Les règles suivantes ont pour la première fois été précisées dans [12] . Elles sont bien développées dans [13]. Voici la liste des règles de réécriture les plus importantes dans les figure :

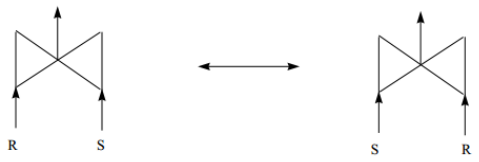


Figure 1.4: Commutativité des jointures [3]

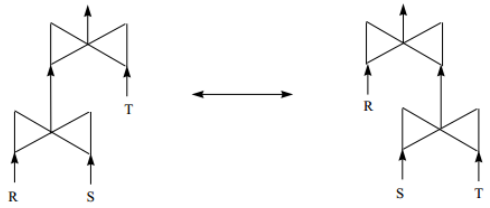


Figure 1.5: Associativité des jointures [3]

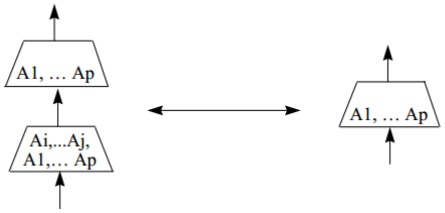


Figure 1.6: Fusion des projections [3]

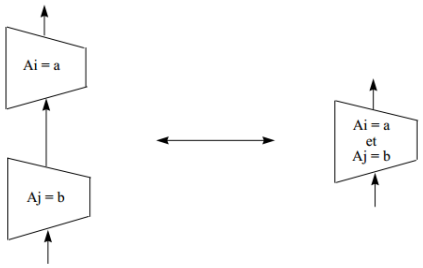


Figure 1.7: Regroupement des restrictions [3]

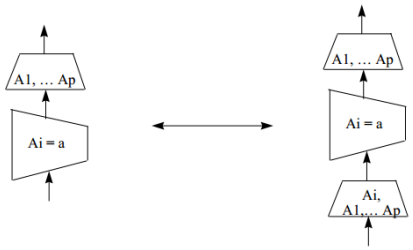


Figure 1.8: Quasi-commutativité des restrictions et des projections [3]

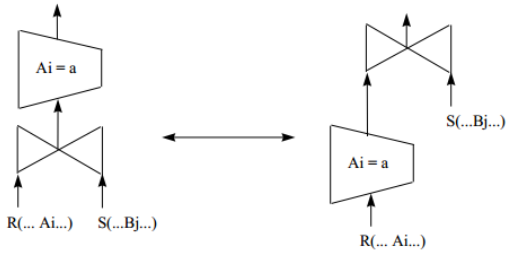


Figure 1.9: Quasi-commutativité des restrictions et des jointures [3]

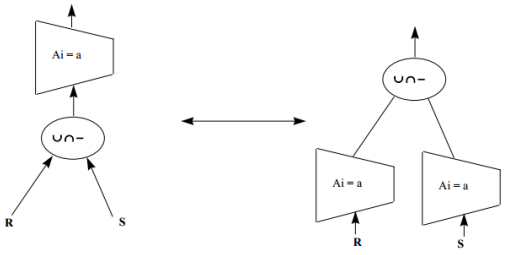


Figure 1.10: Commutativité des restrictions avec les unions, intersections ou différences [3]

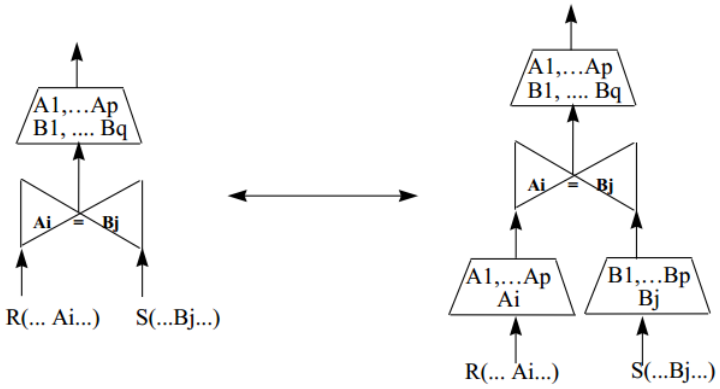


Figure 1.11: Quasi-commutativité des projections et jointures (la projection ne doit pas perdre les attributs de jointure) [3]

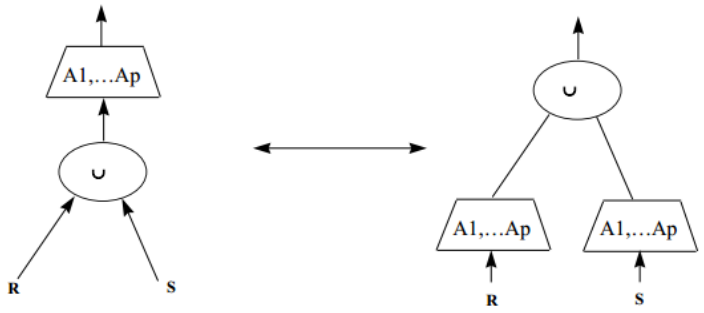


Figure 1.12: Commutativité des projections avec les unions [3]

3.2 Optimisation physique

L'optimisation physique effectue le choix des meilleurs algorithmes pour les opérateurs de bas niveau compte tenu de la taille des données et des chemins d'accès disponibles. L'optimisation logique reste au niveau de l'algèbre relationnelle, alors que l'optimisation physique permet en particulier d'élaborer les annotations. Permet les opérations physique il existe l'opération de jointure, la jointure est l'opération la plus coûteuse et Son optimisation est très importante.

3.2.1 Boucles imbriquées

Est le plus simple. Il consiste à lire séquentiellement la première relation R1 et à comparer chaque tuple lu avec chaque tuple de la deuxième R2. R1 est appelée relation externe et R2 relation interne. Pour chaque tuple de R1, on est donc amené à lire chaque tuple de R2.
L'algorithme est schématisé dans la figure 1.13.

```
Boucles_Imbriquées (R1,A, R2,B){
Pour chaque page B1 de R1 faire{
        Lire (R1,B1) ;
        Pour chaque page B2 de R2 faire{
                Lire (R2,B2) ;
                Pour chaque tuple  Tuple1 de B1 faire
                        Pour chaque tuple Tuple2 de B2 faire{
                                si Tuple1.A = Tuple2.B alors
                                ECRIRE(Résultat, Tuple1 || Tuple2) ;
                        }
                }
}
```

Figure 1.13: Algorithme des boucles imbriquées [3]

3.2.2 Tri-fusion

Le tri d'une relation sur un ou plusieurs attributs utilise l'algorithme de tri-fusion (sort-merge) en mémoire externe. Il est du type "diviser pour régner", avec deux phases. Dans la première phase on décompose le problème récursivement en sous-problèmes, et ce jusqu'à ce que chaque sous-problème puisse être résolu simplement et efficacement. La deuxième phase consiste à

agréger récursivement les solutions. Dans le cas l'algorithme de tri-fusion, les deux phases se résument ainsi:

- Partitionnement de la table en fragments telles que chaque fragment tienne en mémoire centrale. On trie alors chaque fragment (en mémoire), en général avec l'algorithme de Quicksort.

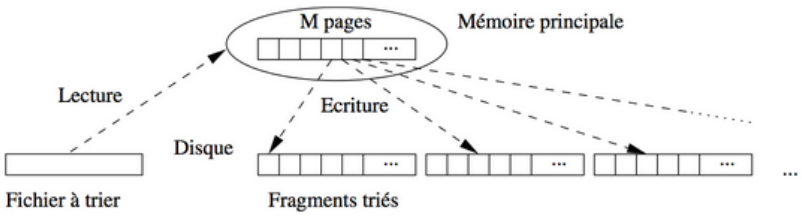


Figure 1.14: Algorithme de tri-fusion: phase de tri [4]

- Fusion des fragments triés.

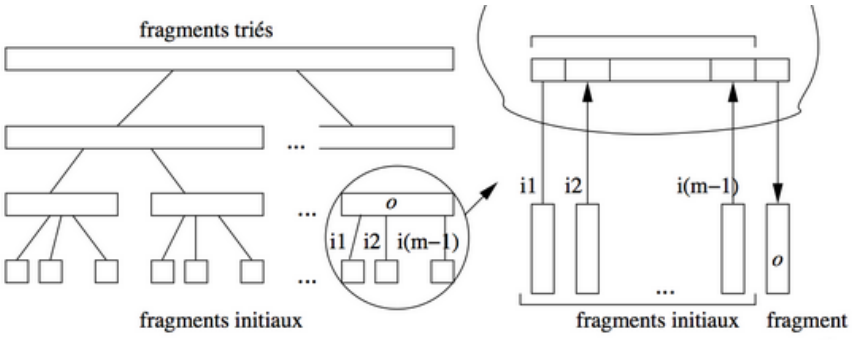


Figure 1.15: 6 Algorithme de tri-fusion: la phase de fusion [4]

3.2.3 Hachage

L'algorithme par hachage le plus simple consiste à hacher seulement la première relation dans un fichier haché si possible gardé en mémoire. Ensuite, on balaye la deuxième relation, et pour chaque tuple lu on tente d'accéder au fichier haché en testant l'existence d'enregistrement de clé identique dans le fichier Tant que l'on trouve des enregistrements de clé similaire, on compose la jointure résultat.

```
Hachage (R1,A, R2,B) {
    Pour chaque page B1 de R1 faire  // hacher R1 sur A{
        Lire (R1,B1) ;
        Pour chaque tuple  Tuple1 de B1faire{
            p = h(Tuple1,A) ;
            ECRIRE (Fichier_Haché, Paquet p,Tuple1) ; }}
    Pour chaque page B2 de R2 faire // Parcourir R2{
        Lire (R2,B2) ;
        Pour chaque tuple  Tuple2 de B2 faire  {
            Tant que PROBE(Fichier_Haché,Tuple2) faire {
                // Joindre si succès
                ACCEDER (Fichier_Haché, Tuple1) ;
                si Tuple1.A = Tuple2.B alors
                    ECRIRE(Résultat,Tuple1‖ Tuple2) ;
                }
            }
        }
}
```

Figure 1.16: Algorithme par hachage [3]

4 La Recherche du meilleur plan

L'optimisation physique permet de prendre en compte le modèle de coût afin de déterminer le meilleur plan, ou un plan proche de celui-là. Elle part d'un arbre en forme canonique, par exemple composé de sélections dont les critères sont sous forme normale conjonctive, puis de jointures, unions, intersections et différences, suivis encore d'éventuelles sélections/projections finales, certaines de ces opérations pouvant être absentes. Elle transforme cet arbre en un plan d'accès en choisissant les algorithmes pour chaque opérateur, et en modifiant éventuellement l'arbre afin de profiter de propriétés physiques de la base (index par exemple).

le nombre de plans d'exécution possible peut être très grand pour des requête complexes. L'ensemble des plans possible est appelé espace des plans. Afin d'éviter d'explorer exhaustivement cet espace de recherche, c'est-à-dire d'explorer tous les plans, les optimiseurs modernes sont construits comme des générateurs de plans couplés à une stratégie de recherche découlant des techniques d'optimisation de la recherche opérationnelle [14].

Un optimiseur est fermé lorsque tous les opérateurs et algorithmes d'accès, ainsi que toutes les règles de permutation de ces opérateurs, sont connus à la construction du système. Les systèmes relationnels purs ont généralement des optimiseurs fermés. La stratégie de recherche dans les optimiseurs fermés est basée sur un algorithme déterministe, qui construit une solution pas à pas soit en appliquant une heuristique ou une méthode d'évaluation progressive permettant d'exhiber le meilleur plan .

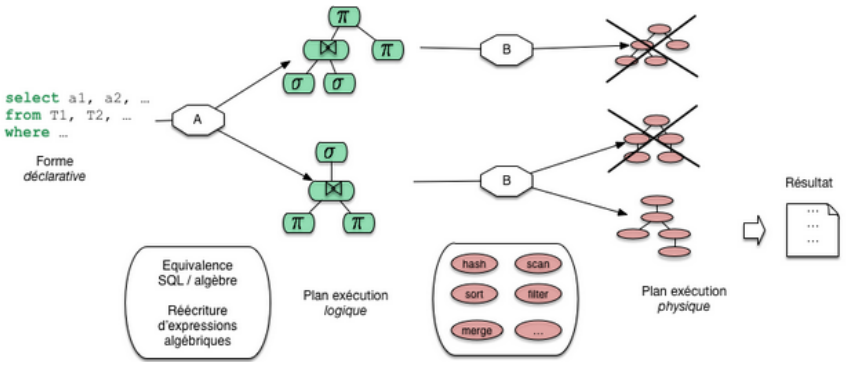


Figure 1.17: Processus général d'optimisation [4]

5 Stratégies de recherche

Le rôle d'une stratégie de recherche est d'exploiter l'ensemble des plans d'exécution candidats afin d'en trouver celui de coût optimum en utilisant les modèles de coût. Intuitivement, il est préférable de vérifier tous les plans d'exécution dans l'espace de recherche afin d'assurer que le plan choisi pour l'évaluation de requête est le meilleur en terme de coût d'exécution. Il s'agit donc d'un algorithme exhaustif qui essaie toutes les combinaisons de règles de transformation pour générer tous les plans candidats possibles pour la requête donnée. Pour chaque plan candidat, son coût d'exécution est calculé grâce au modèle de coût et le plan ayant le coût minimal est choisi pour évaluer la requête.

Cependant, pour les requêtes complexes composées d'un grand nombre d'opérations, le nombre de plan candidats peut être immense et l'application de l'algorithme exhaustif s'avère très coûteux. Le temps d'optimisation d'une requête est alors inacceptable. Pour éviter d'explorer tout l'espace de recherche, l'optimiseur utilise une stratégie de recherche issue de la métaheuristique, afin de n'explorer qu'une partie de l'espace et de trouver un plan candidat optimal selon le critère de la stratégie utilisée.

Les métaheuristiques sont des méthodes génériques qui peuvent optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé.

L'objectif des métaheuristiques est de résoudre un problème d'optimisation donné : Elle cherche un objet mathématique (une permutation, un vecteur, etc.) minimisant (ou maximisant) une fonction objectif, qui décrit la qualité d'une solution au problème.L'ensemble des solutions possibles forme l'espace de recherche.

Les métaheuristiques manipulent une ou plusieurs solutions, à la recherche d'un optimum, une meilleure solution au problème. Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à une solution optimale.

L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti ou en une précision demandée. Il faut noter que la solution trouvée par l'algorithme est souvent une solution optimale locale qui est plus ou mois proche de la solution optimale selon la stratégie.

Dans le domaine d'optimisation de requêtes, certains algorithmes issus de métaheuristiques sont couramment utilisés. Ce sont des algorithmes de programmation dynamique, incrémental et recuit simulé, algorithme Génétique ce dernier fait l'objet de notre étude . Nous résumons ces algorithmes comme suit :

5.1 Programmation dynamique (DP)

La programmation dynamique est une méthode très connue en recherche opérationnelle, dont le principe est que toute sous-politique d'une politique optimale est optimale. Ainsi, si deux sous-plans équivalents du plan optimal cherché sont produits, il suffit de garder le meilleur et de le compléter pour

atteindre le plan optimal. Cette technique, employée dans le système R [9] pour des plans sans produit cartésien, permet de construire progressivement le plan, par ajouts successifs d'opérateurs. L'algorithme commence par créer tous les plans monorelation, et construit des plans de plus en plus larges étape par étape. A chaque étape, on étend les plans produits à l'étape précédente en ajoutant un opérateur. Quand un nouveau plan est généré, la collection de plans produits est consultée pour retrouver un plan équivalent. Si un tel plan est trouvé, alors les coûts des deux plans sont comparés et seul celui de coût minimal est conservé.

5.2 Recuit simulé

procède à partir d'un plan que l'on tente d'optimiser en appliquant des transformations successives. Les transformations retenues améliorent ce plan exceptées quelques unes introduites afin d'explorer un espace plus large avec une probabilité variant comme la température du .La température est de fait une variable dont la valeur est très élevée au départ et qui diminue au fur et à mesure des transformations, de sorte à ne plus accepter de transformation détériorant le coût quand le système est gelé. Cette stratégie, bien connue en recherche opérationnelle, permet de converger vers un plan proche de l'optimum, en évitant les minimums locaux.

5.3 Algorithme Génétique

L'algorithme génétique est un algorithme d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle : croisements, mutations, sélection. Un algorithme génétique recherche le ou les extrema d'une fonction dé nie sur un espace de données. Le déroulement d'un algorithme génétique peut être découpé en cinq parties :

1. La création de la population initiale
2. L'évaluation des individus
3. La création de nouveaux individus
4. L'insertion des nouveaux individus dans la population
5. Itération du processus

6 Conclusion

Ce chapitre, nous a permis de traiter les notions de base de l'optimisation d'une requête, et comment l'optimiseur d'un SGBD fait le traitement pour répondre à une requête utilisateur, et qui passe par de choix d'une stratégie pour sélectionner un plan optimal. Plusieurs techniques ont été proposées en l'occurrence des algorithmes génétiques qui sera abordé dans le chapitre suivant.

Chapitre 2

Les algorithmes génétiques

1 Introduction

Les algorithmes évolutifs (Evolutionary Algorithms) sont des techniques de recherche inspirées de l'évolution biologique des espèces, apparues à la fin des années 1950 [15].

Les algorithmes génétiques (AG) appartiennent à la famille des algorithmes évolutionnistes (un sous-ensemble des métaheuristiques [16]). Leur but est d'obtenir une solution approchée, en un temps correct, à un problème d'optimisation, lorsqu'il n'existe pas (ou qu'on ne connaît pas) de méthode exacte pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle développée au XIX$^{\text{ème}}$ siècle par le scientifique Darwin [17] et l'appliquent à une population de solutions potentielles au problème donné.

Les algorithmes génétiques ont la particularité de s'inspirer de l'évolution des espèces dans leur cadre naturel. Les espèces s'adaptent à leur cadre de vie qui peut évoluer, les individus de chaque espèce se reproduisent, créant ainsi de nouveaux individus [18] , certains subissent des modifications de leur ADN, certains disparaissent....

Un algorithme génétique va reproduire ce modèle d'évolution dans le but de trouver des solutions pour un problème donné. On fera usage, alors, de termes empruntés au monde des biologistes et des généticiens et ceci afin de mieux représenter chacun des concepts abordés :

1. Dans notre cas, une population sera un ensemble d'individus.
2. Un individu sera une réponse à un problème donné.
3. Un gène sera une partie d'une réponse au problème, donc d'un individu.
4. Une génération est une itération de notre algorithme.

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.

2 Les Concepts importants des algorithmes génétiques

La transposition des concepts biologiques dans un cadre artificiel a conduit à la définition des concepts suivants :

Les chromosomes (individus) : sont les éléments à partir des quels sont élabores les solutions à un problème posé (représente une solution potentielle).
Gène : bit ou ensemble de bits codant une information.
Fitness (coût) : fonction à optimiser, mesure d'efficacité des individus solutions, régissant les transformations génétiques appliquées.et aussi appelé une (fonction d'adaptation).
Population : ensemble d'individus d'une même génération (espace de recherche).

3 Fonctionnement d'un algorithme génétique

Les algorithmes génétiques sont des algorithmes itératifs qui suivent le schéma des évolutionnistes. Leur fonctionnement est schématisé dans la figure 2.1 dont les étapes sont :

1. A l'initialisation de l'algorithme, une population de taille fixée est générée aléatoirement.

2. Pour la génération k , on reproduit une partie de la population en fonction de l'adaptation de chaque individu : plus la fitness (relative, i.e. par rapport à celle des autres individus) d'un individu est élevée (basse), plus il sera reproduit dans la nouvelle population
3. On insère les nouveaux individus dans la population
4. On applique ensuite les différents opérateurs (mutation, croisement ....)
5. On réitère ces opérations à partir de la 2ème étape jusqu'à ce qu'un critère d'arrêt soit satisfait. Différentes critères d'arrêt de l'algorithme génétique peuvent être choisis : nombre de générations fixé (temps constant), convergence de la population, population n'évoluant plus suffisamment...

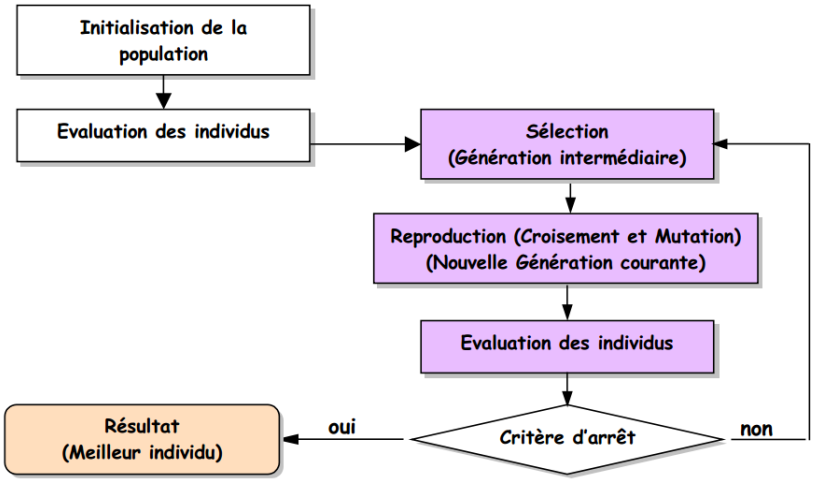


Figure 2.1: Fonctionnement d'un Algorithme génétique

4 Caractéristiques des algorithmes génétiques

4.1 Codage des données

Le premier pas dans l'implémentation des algorithmes génétiques est de créer une population d'individus initiaux. En effet, les algorithmes génétiques

agissent sur une population d'individus, et non pas sur un individu isolé. Par analogie avec la biologie, chaque individu de la population est codé par un chromosome ou génotype. Une population est donc un ensemble de chromosomes. Chaque chromosome code un point de l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome.

Représentation binaire

Chaque gène dispose du même alphabet binaire 0, 1 Un gène est alors représenté par un entier long (32 bits), les chromosomes qui sont des suites de gènes sont représentés par des tableaux de gènes et les individus de notre espace de recherche sont représentés par des tableaux de chromosomes.

Représentation avec réels

Ce type de codage le gène est une valeur réelle directement liée au problème.

Représentation à l'aide d'arbres syntaxique

Ce type de codage utilise une structure arborescente. Un arbre syntaxique est un arbre contenant deux types de nœuds [19] :

1. les nœuds internes ou symboles non terminaux .
2. les feuilles ou symboles terminaux .

Finalement, le choix du type de codage ne peut pas être effectué de manière sûre dans l'état actuel des connaissances. Selon les chercheurs dans ce domaine, la méthode actuelle à appliquer dans le choix du codage consiste à choisir celui qui semble le plus naturel en fonction du problème à traiter et développer ensuite l'algorithme de traitement [20],[21].

4.2 La fonction d'évaluation

On appelle aussi fonction objectif ou fonction d'adaptation ou fonction fitness, associe une valeur de performance à chaque individu ce qui offre la possibilité de le comparer à d'autres individus et permet à l'algorithme génétique de déterminer qu'un individu sera sélectionné pour être reproduit ou pour déterminer s'il sera remplacé [22]. La fonction d'adaptation est un élément fondamental lors de la modélisation d'un AG.

5 Les opérateurs génétiques

Les opérateurs génétiques permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. Ces opérateurs sont la base des Algorithmes Génétiques, et le plus souvent sont : sélection, croisement et mutation. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'états.

5.1 L'opérateur de sélection

Cet opérateur est chargé de définir quels seront les individus de P qui vont être dupliqués dans la nouvelle population P' et vont servir de parents (application de l'opérateur de croisement). Soit n le nombre d'individus de P, on doit en sélectionner n/2 (l'opérateur de croisement nous permet de repasser à n individus). Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population. On trouve essentiellement quatre types de méthodes de sélection différentes :

- La méthode de la ”loterie biaisée” (roulette wheel) de GoldBerg,
- La méthode ”élitiste”,
- La sélection par tournois,
- La sélection universelle stochastique.

5.1.1 La loterie biaisée ou roulette wheel

Cette méthode est la plus connue et la plus utilisée. Avec cette méthode chaque individu a une chance d'être sélectionné proportionnelle à sa performance, donc plus les individus sont adaptés au problème, plus ils ont de chances d'être sélectionnés. Pour utiliser l'image de la ”roue du forain”, chaque individu se voit attribué un secteur dont l'angle est proportionnel à son adaptation, sa ”fitness”. On fait tourner la roue et quand elle cesse de tourner on sélectionne l'individu correspondant au secteur désigné par une sorte de ”curseur”, curseur qui pointe sur un secteur particulier de celle-ci après qu'elle se soit arrêté de tourner.

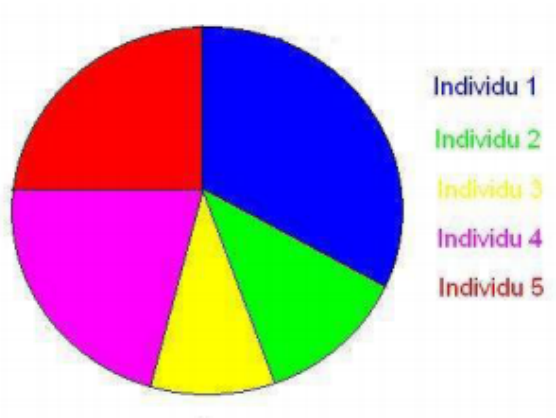


Figure 2.2: Schéma d'une roulette [5]

5.1.2 Méthode élitiste

Cette méthode consiste à sélectionner les n individus dont on a besoin pour la nouvelle génération P' en prenant les n meilleurs individus de la population P après l'avoir triée de manière décroissante selon la fitness de ses individus. Il est inutile de préciser que cette méthode est encore pire que celle de la loterie biaisée dans le sens où elle amènera à une convergence prématurée encore plus rapidement et surtout de manière encore plus sûre que la méthode de sélection de la loterie biaisée ; en effet, la pression de la sélection est trop forte, la variance nulle et la diversité inexistante, du moins le peu de diversité qu'il pourrait y avoir ne résultera pas de la sélection mais plutôt du croisement et des mutations. Là aussi il faut opter pour une autre méthode de sélection.

5.1.3 La sélection par tournois

Cette méthode est celle avec laquelle on obtient les résultats les plus satisfaisants. Le principe de cette méthode est le suivant : on effectue un tirage avec remise de deux individus de P, et on les fait "combattre". Celui qui a la fitness la plus élevée l'emporte avec une probabilité p comprise entre 0.5 et 1. On répète ce processus n fois de manière a obtenir les n individus de P' qui serviront de parents. La variance de cette méthode est élevée et le fait d'augmenter ou de diminuer la valeur de p permet respectivement de diminuer ou d'augementer la pression de la sélection.

5.1.4 La sélection universelle stochastique

Cette méthode semble être très peu utilisée et qui plus est possède une variance faible, donc introduit peu de diversité, nous n'entrerons donc pas dans les détails, on se contentera d'exposer sa mise en oeuvre : On prend l'image d'un segment découpé en autant de sous-segments qu'il y a d'individus. Les

individus sélectionnés sont désignés par un ensemble de points équidistants.

5.2 L'opérateur de croisement ou crossover

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. La figure 2.3 décrit le croisement en un point . Il est tout à fait possible de faire des croisements aléatoires. Toutefois, une solution largement utilisée est d'effectuer des croisements multi-points (figure 2.4).

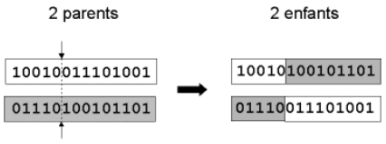


Figure 2.3: Croisement en un point d'un individu de 14 bits [5]

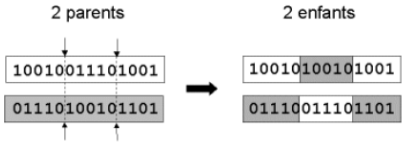


Figure 2.4: Croisement en deux point d'un individu de 14 bits [5]

Dans le schéma de croisement uniforme (figure 2.5), les bits de la chaîne sont comparés entre les deux parents. Les bits sont échangés avec une probabilité fixe, en général 0.5. Sa probabilité d'apparition est un paramètre de l'algorithme génétique et dépend du problème et de la technique de recombinaison. La probabilité d'un croisement est alors comprise entre 0 et 1 strictement.

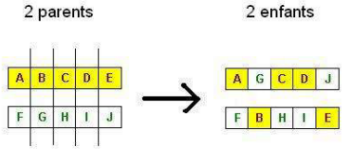


Figure 2.5: Croisement uniforme [5]

5.3 L'opérateur de mutation

L'opérateur de mutation effectue en général un déplacement local sur la solution représentée par un individu. La mutation est considérée comme un opérateur marginal pour les AGs, bien qu'elle leur confère la propriété d'ergodicité, c'est-à-dire que tous les points de l'espace de recherche peuvent être atteints ; cet opérateur a une grande importance, de point de vue théorique. L'opérateur classique de mutation sur les chaînes de bits choisit aléatoirement une position dans le gène et le remplace par son complémentaire pour construire l'individu mutant comme l'illustre la figure 2.6.

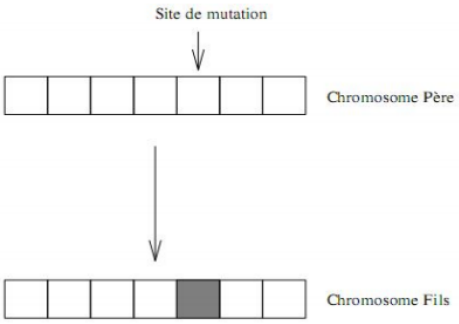


Figure 2.6: Mutation classique [6]

On peut généraliser cette mutation sur des chaînes à domaines discrets de tailles quelconque en changeant la valeur du gène choisi par une autre de son domaine, proche de la valeur initiale. Il existe également un principe de mutation adaptative figure 2.7, permettant d'optimiser le taux

de mutation en codant ce dernier dans la structure du chromosome[23] . Ce second chromosome est géré de la même manière que le premier chromosome codant l'espace d'état, c'est-à-dire lui-même soumit aux opérateurs génétiques (croisement et mutation). Au cours du déroulement de l'algorithme, les gènes et les individus ayant des probabilités de mutation élevées auront tendance à disparaître à mes ure que la population converge vers l'optimum. De même, les gènes ayant des probabilités de mutation trop faibles ne peuvent évoluer favorablement et tendent à être supplantés.

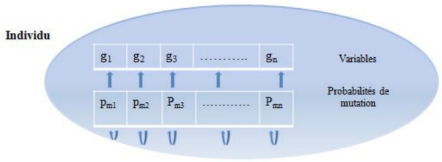


Figure 2.7: Mutation adaptative [6]

6 Conclusion

Les Algorithmes Génétiques sont les approches métaheuristiques les plus répandues pour résoudre des problèmes difficiles d'optimisations et de recherche. Leur efficacité est déterminée par les opérateurs qui sont utilisés et par la fonction d'évaluation. Le chapitre suivant consiste en l'application de cet algorithme pour tester la performance du choix d'un plan optimum.

Chapitre 3

Présentation et implémentation de notre approche

1 Introduction

Les algorithmes génétiques ont montré leur efficacité dans la résolution de nombreux problèmes et notamment dans les problèmes d'optimisation. Dans cette partie, nous présentons les étapes et les différents notions des algorithmes génétiques et les paramètres de l'environnement de l'optimisation des requêtes. Pour cela, on prend un exemple de requête de sept (07) relations afin d'implémenter notre approche sous l'environnement JAVA, et l'estimation de cout d'exécution sous Microsoft SQL server. Le résultat obtenu en appliquant les différents opérateurs génétiques pour trouver un plan optimal sera discuté.

2 Les outils utilisés

Java et l'éditeur netbeans :

Java est à la fois un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.
Il est à la fois un langage de programmation et un environnement d'exécution. Et sa particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modifications... C'est la plate-forme qui garantit la portabilité des applications développées en Java.
Netbeans est un environnement de développement intégré (EDL),placé en open source par sun en juin 2000 sous licence CDDL (common development

and distribution license).En plus de java,Netbeans permet également de supporter différents qutres langages ,comme python ,C,C++,XML et HTML.Il comprend toutes les caractéristique d'un IDE moderne (éditeur en couleur ,projet multi-langage ,refactoring ,éditeur graphique d'interface et de pages web).
Netbeans est disponible sous windows , linux ,solaris(sur x86 et SPARC),Mac OS X et open VMS.

SQL Server :

Microsoft SQL Server est un système de gestion de base de données (abrégé en SGBD) incorporant entre autres un SGBDR (SGBD relationnel) développé et commercialisé par la société Microsoft. Il fonctionne sous les OS Windows, Linux, Mac OS via docker; une version docker existe en téléchargement sur le site de microsoft.

- Le moteur relationnel (OLTP) appelé SQL Server ,
- Le moteur décisionnel (OLAP) appelé SSAS (SQL Server Analysis Services) incluant un moteur de stockage pour les cubes, des algorithmes de forage (data mining) et différents outils de BI (Business Intelligence) ,
- Un ETL (Extract Transform and Load) appelé SSIS (SQL Server Integration Services) destiné à la mise en place de logiques de flux de données, notamment pour alimenter des entrepôts de données (data warehouse) ,
- Un outil de génération d'état appelé SSRS (SQL Server Reporting Services) permettant de produire des rapports sous différentes formes et exploitant les ressources du moteur décisionnel (bases "resportServer...") à la fois pour y stocker les rapports mais aussi y cacher les données de ces derniers afin de faire du "warmup" ,

- Un système de planification de travaux et de gestion d'alerte appelé Agent SQL qui utilise lui aussi les services du moteur SQL (base msdb).

Exemple :

Pour notre approche, nous avons choisi l'exemple ci-dessus pour tester l'efficacité des AG :

Order_items (sn , order_id , car_id , Price)
Orders (order_id , company_id , contact_id)
Contacts(contact_id ,name , company_id)
Companies(company_id , name ,location)
brands(brand_id , brand , parent_id)
Colors(color_id , color)
Cars(car_id ,color_id , brand_id)

La Requête utilisée ,est exprimée en SQL comme suit :

```
SELECT
    orders.order_id as'Order ID' ,
    companies.name as 'Company Name',
    contacts.name as  'Contact Name',
    CONCAT( brands.brand ,',',colors.color) as  'Car Ordered' ,
    order_items.price as 'price'
    FROM order_items
    left outer join orders on orders.order_id=order_items.order_id
    left outer join contacts on orders.contact_id = contacts.contact_id
    left outer join on orders.company_id=companies.company_id
    left outer join cars on order_items.car_id =cars.car_id
    left outer joinon cars.color_id = colors.color_id
    left outer join brands on cars.brand_id = brands.brand_id
    order by brands.brand asc
```

En appliquant les différentes règles de réécritures et les propriétés des différents opérateurs algébriques .On peut avoir plusieurs alternatives des plans d'exécutions :

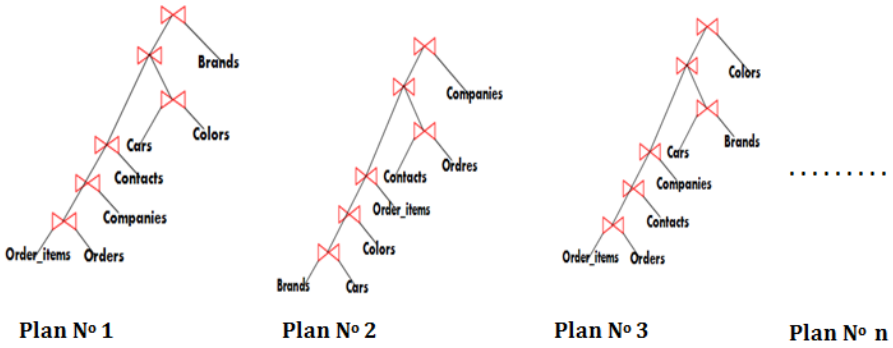


Figure 3.1: Les plan d'exécutions équivalents

3 Description de l'AG d'optimisation de requête

3.1 L'Algorithme génétique d'optimisation de requête

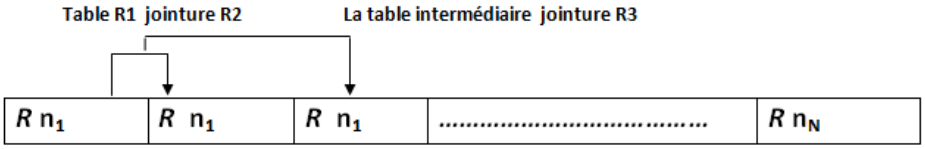
Début
T :=0
Répéter
Générer la population initiale
Effectuer le codage
Évaluation de la fonction de fitness
Fait
Appliquer les opérateurs génétiques
T :=T+1
Jusqu' à arrêt
Fin

3.2 La génération de population initiale

Le principe de fonctionnement de l'algorithme génétique se base sur la création d'une population initiale qui représente l'espace de recherche qui contient des membres représente l'arbre algébriques correspondant à l'expression d'un requête utilisateur.

Le codage des individus est une étape de modélisation fondamentale dans les algorithmes génétiques, il permet de représenter les données, les paramètres et les solutions dans notre cas nous avons choisi la représentation sous forme d'un vecteur de dimension N . Chaque élément de vecteur représente une relation / table. Un gène représente les relations ou bien les table.

Comme notre étude se focalise sur l'opérateur jointure seulement parce que cette opérateur est le plus couteuse ou exprimé dans le vecteur des géne . Les relations ayant un lien de jointure dans l'ordre adéquate. En plus, chaque



gène doit contenir la liste des clés primaires et les clés étrangères, et que chaque chromosome a l'information concerne le coût d'exécution

3.3 La fonction d'adaptation(Fitness)

L'utilisation d'algorithme génétique nécessite une fonction d'évaluation, cette fonction permet de mesurer la performance d'un individu dans la résolution d'un problème posé. la fonction à prendre comme f (x) est : f = coût

3.4 Les opérateurs génétiques

3.4.1 Sélection:

L'opérateur de sélection permet de sélectionner les meilleurs individus dans la population pour participer dans la génération de la prochaine population. Nous avons utilisé la sélection par tournoi ou on sélectionne les parents qui ont le minimum de Fitness. ce type de sélection retourne les meilleurs individus de la population.

3.4.2 Croisement:

L'opérateur de croisement permet à deux individus d'échanger leurs gènes En vue de créer de nouveaux individus plus intéressants. Le croisement est appliqué sur deux individus pères choisis par l'opérateur de sélection. Dans notre approche on utilise un croisement uniforme, dans le but d'exploiter au mieux la distribution des plans d'exécution.

3.4.3 Mutation :

L'opérateur de mutation permet de modifier occasionnellement des gènes d'un individu pour permettre d'explorer certaines zones dans la codification des individus ou le croisement ne peut pas l'explorer . Nous avons utiliser l'opérateur de mutation le plus simple ,c'est la mutation discrete qui introduise des nouvelles propriétés dans la population.

Après l'application des opérateurs génétiques les opérations : de croisement ont généré 2C nouveaux individus , du mutation ont généré M nouveaux individus donc on a N+2C+M membres , l'algorithme élimine le pire Membre, à partir d'une fonction « fitness » (le pire plan).

4 Implémentation de notre approche

I' implémentation de cette approche est faite sous le langage java et le SGBD Sql Server 2008 pour calculer le coût de chaque plan d'exécution. Le coût de 1er plan d'exécution est comme suite :

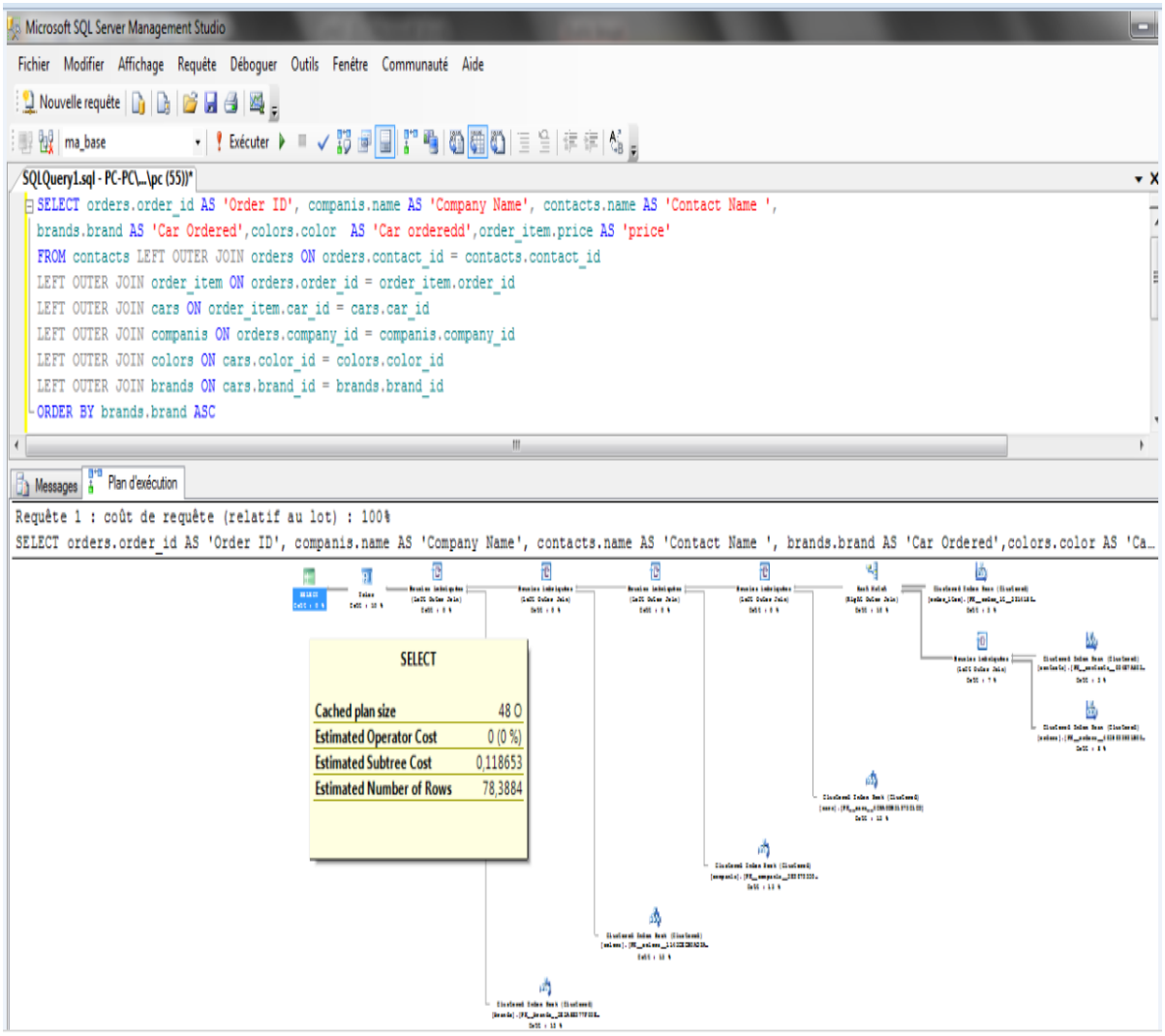


Figure 3.2: Estimation de coût d'exécution

La 1er étape consiste à choisi aléatoirement quatre(04) plans correspondant à la requête ce dessus avec leurs coût tel que chaque relation est identifiée dans le vecteur plan par sa clé primaire (identifiant):

P1 : coût = 0.118653

Contact_id	Order_id	sn	Car_id	Company_id	Color_id	Brand_id

P2 : coût = 0.161689

Car_id	sn	Color_id	Order_id	Contact_id	Brand_id	Company_id

P3 : coût = 0.1656785

Car_id	sn	Order_id	Contact_id	Color_id	Brand_id	Company_id

P4 : coût = 0.160629

Car_id	Color_id	sn	Order_id	Contact_id	Company_id	Brand_id

Les quatre(04) plan représentent l'ensemble des chromosomes de la population initiale de l'A.G qui seront introduit dans notre application afin de générer d'autre population (voir la figure 3.3)

Figure 3.3: 1er itération

On remarque que la 1er itération, va produire d'autre plans après l'application des différent opérateurs, tel que le P3 sera ignoré et remplacé par un autre plan performant. le déroulement de l'itération est comme suit :

1. A chaque fois, on reproduit une partie de la population en minimisant le cout de chaque chromosome : plus la fitness d'un chromosome est basse, plus il sera reproduit dans la nouvelle population
2. On insère les nouveaux individus dans la population

3. Avec l'application des différents opérateurs d'algorithme génétique en plusieurs itérations

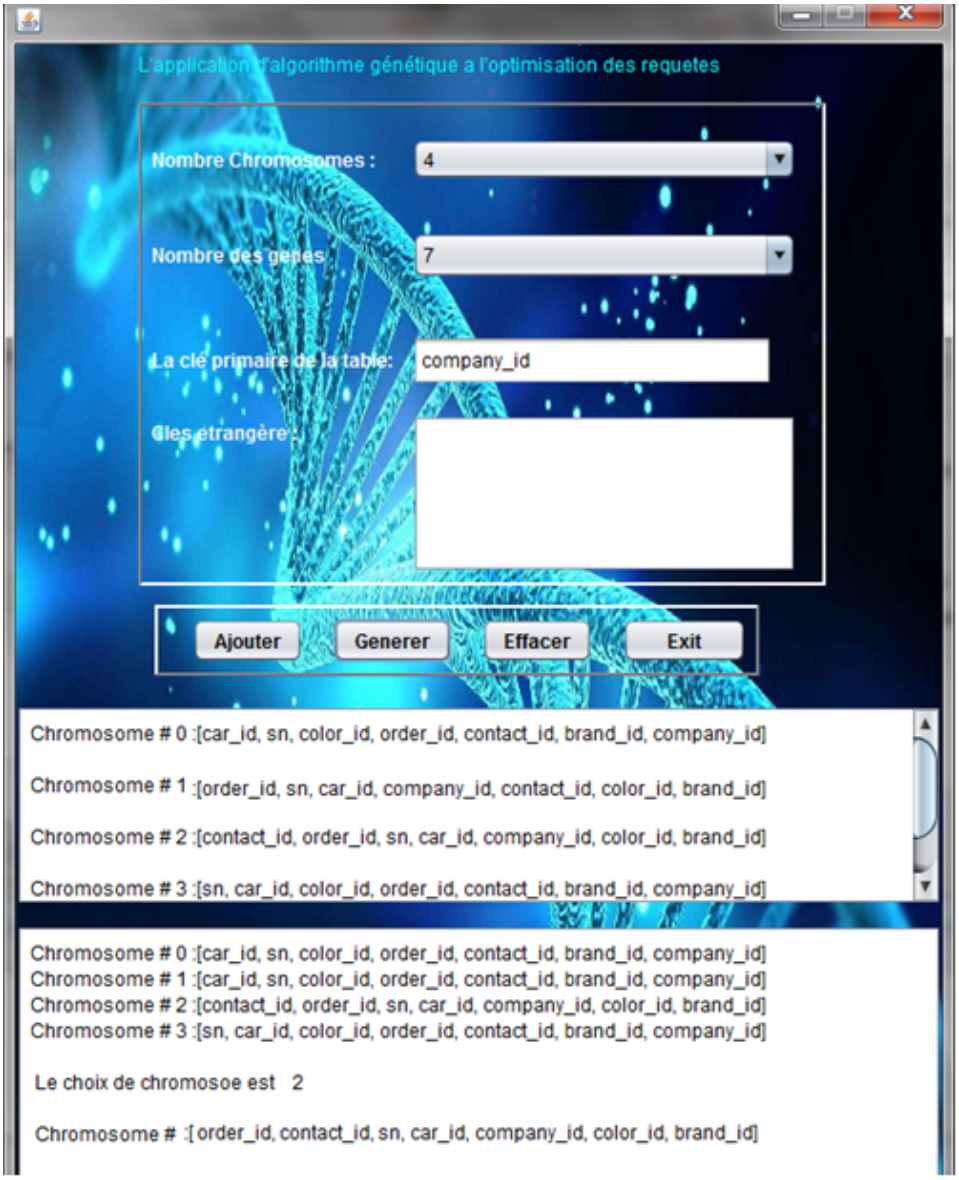


Figure 3.4: Dernier itération

4. On aura une population avec moindre coût (figure 3.4):

La population finale après plusieurs itérations est comme suite :

P37 : coût = 0.118653

Contact_id	Order_id	sn	Car_id	Company_id	Color_id	Brand_id

P38 : coût = 0.092433

Sn	Car_id	Color_id	Order_id	Contact_id	Brand_id	Company_id

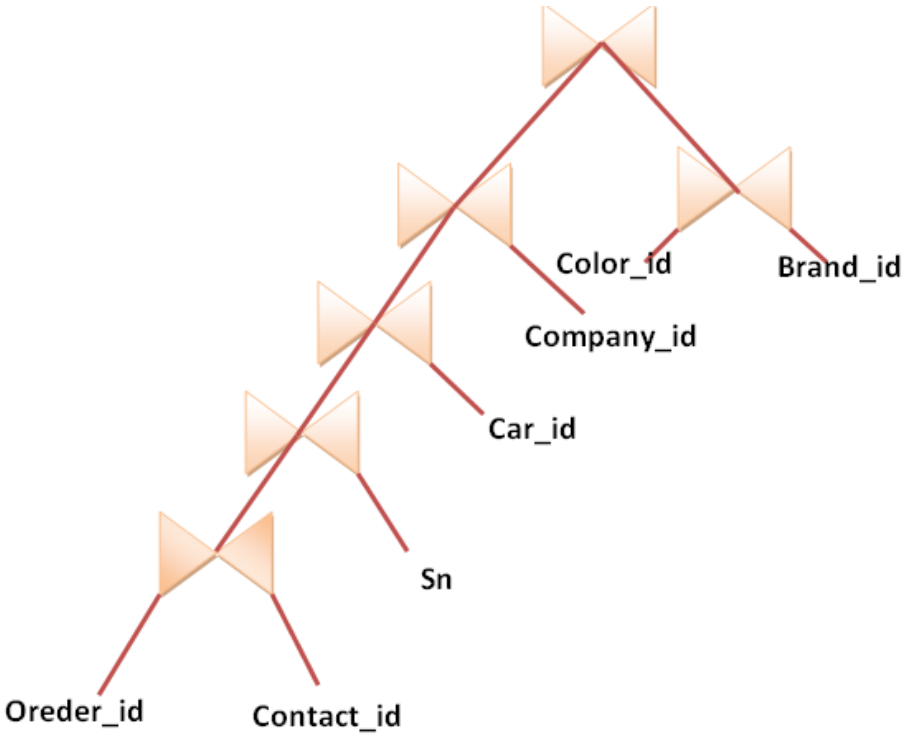
P39 : coût = 0.0933731

Order_id	sn	Car_id	Company_id	Contact_id	Color_id	Brand_id

P40 : coût = 0.0900734

Order_id	Contact_id	Sn	car_id	Company_id	Color_id	Brand_id

On observe que P40 a un cout minimal et donc l'arborescence correspondante:



5 Conclusion

Dans ce chapitre, nous avons présenté une démarche d'application des algorithme génétiques qui se base sur les opérateur de croisement, mutation et sélection dans le but de choisir un plan d'exécution optimal.

Nous avons décrit toutes les étapes de notre démarche à travers un exemple de requête de jointure.Sous l'implémentation avec le langage JAVA le SGBD SQL Server, et le discussion des résultat obtenus pas à pas, jusqu'à la dernier itération.

En fin, le résultat de notre expérimentation s'avère intéressant vu que le coût de plan d'exécution est le plan optimal.

Conclusion générale et perspective

L'optimisation des requêtes est un problème difficile et plusieurs techniques et méta heuristiques ont été proposés pour améliorer l'optimiseur des SGBD.

L'objectif que nous avons visé lors de ce travail est l'application d'un algorithme génétique pour l'optimisation de l'exécution d'une requête par la sélection d'un plan optimal. Ce choix est justifié par sa puissance et son efficacité, en termes d'optimisation.

Pour ce faire, nous avons commencé par l'introduction des différents données nécessaire à savoir la requête utilisateur et la population initiale, puis l'application en ordre des opérateurs de l'algorithme génétique qui nous produire dans la dernière itération un plan performant. Nous avons construit les interfaces graphiques permettant de réagir avec cette outil.

Ce travail nous a permis de se familiariser avec le langage JAVA, d'approfondir nos connaissances dans le domaine de l'optimisation des bases de données en particulier le module optimiseur d'un SGBD dans un environnement SQL server et d'acquérir des connaissances sur l'algorithme génétique.

En guise de perspectives futures, notre approche peut être améliorée et complétée par la comparaison des résultats avec d'autre technique à savoir la colonie de fourmis.

Bibliographie

[1] Tianxiao Liu. *Proposition d'un cadre générique d'optimisation de requêtes dans les environnements hétérogènes et répartis*. PhD thesis, Université de Cergy Pontoise, 2011. x, 7

[2] Evaluation et optimisation, . URL `http://sys.bdpedia.fr/optim.html`. x, 10, 11

[3] Georges Gardarin. Base de données. *Éd. Eyrolles*, 2003. x, 12, 13, 14, 15, 17

[4] Opérateurs et algorithmes, . URL `http://sys.bdpedia.fr/opalgo.html`. x, 16, 19

[5] Sarra Bouallagui. *Techniques d'optimisation déterministe et stochastique pour la résolution de problèmes difficiles en cryptologie*. PhD thesis, INSA de Rouen, 2010. xi, 29, 31, 32

[6] Elhoussaine Ziyati. Optimisation de requêtes olap en entrepôts de données: Approche basée sur la fragmentation génétique. 2010. xi, 32, 33

[7] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Computing surveys (CsUR)*, 16(2):111–152, 1984. 5

[8] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2):73–169, 1993. 5

[9] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979. 5, 21

[10] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 8

[11] Jason McHugh and Jennifer Widom. Query optimization for xml. Technical report, Stanford InfoLab, 1999. 8

[12] John Miles Smith and Philip Yen-Tang Chang. Optimizing the performance of a relational algebra database interface. *Communications of the ACM*, 18(10):568–579, 1975. 11

[13] JD Ullman. Priciples of database and knowledgebase systems. rockville, maryland, 1988. 11

[14] Arun Swami and Anoop Gupta. *Optimization of large join queries*, volume 17. ACM, 1988. 18

[15] DE Goldberg. 'genetic algorithms in search, optimization & machine learning', addison- wesley publishing company, inc., 1989. 24

[16] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009. 24

[17] Charles Darwin. On the origin ofspecies by means ofnatural selection. *London: Murray*, 1859. 24

[18] John Holland. Adaptation in artificial and natural systems. *Ann Arbor: The University of Michigan Press*, 1975. 24

[19] Nathalie Aussenac-Gilles. Rapport sur les activités antérieures. 2010. 27

[20] David Edward Goldberg and Vincent Corruble. *Algorithmes génétiques: exploration, optimisation et apprentissage automatique*. Ed. Addison-Wesley France, 1994. 27

[21] Jean-Michel Renders. *Algorithmes génétiques et réseaux de neurones*. Hermès, 1994. 27

[22] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003. 28

[23] Thomas Bäck. Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press, 1992. 33