N° d'ordre : 12/2007- E/IN

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie HOUARI BOUMEDIENE
FACULTE DE D'ELECTRONIQUE ET D'INFORMATIQUE

THESE

Présentée pour l'obtention du diplôme de Doctorat d'Etat

En : Informatique
Spécialité : Informatique
Par : Abdelmadjid BOUKRA

Thème

MATERIALISATION DES VUES DANS UN ENTREPÔT DE DONNEES

Soutenue le 30/12/2007, devant le jury composé de :

Mme A. MOKHTARI	Professeur	USTHB	Président
Mr M. AHMED-NACER	Professeur	USTHB	Directeur de thèse
Mme M. BOUKALA	Professeur	USTHB	Examinateur
Mme Z. BOUFAIDA	Professeur	Université de Constantine	Examinateur
Mr M. BOUFAIDA	Professeur	Université de Constantine	Examinateur

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué de manière directe ou indirecte à l'aboutissement de ce travail :

En premier lieu, je remercie vivement Mr Ahmed Nacer Mohamed Professeur à USTHB pour m'avoir accueilli au sein de son équipe, et m'avoir permis d'effectuer cette thèse sous sa direction.

Je tiens à remercier Mr Bouroubi Sadek Maître de conférence à USTHB pour ses fructueux conseils et son aide.

Je remercie également Melle Hassas Salima Professeur à l'université Claude Bernard Lyon 1 pour m'avoir accueilli dans son laboratoire pendant mon stage et m'avoir encouragé.

Comme je n'oublie pas de remercier tous les membres du laboratoire LSI dirigé par Professeur Badache Nadjib pour leurs fructueuses discussions pendants les séminaires et spécialement Mr Daoudi Mourad.

Je ne remercierai jamais assez mon père et ma mère pour tout ce qu'ils ont fait pour moi

Je remercie aussi ma femme qui m'a encouragé et soutenu pendant les moments difficiles.

Résumé

Les entreprises actuelles ont atteint un tel degré de compétition et de concurrence que la prise de décision est devenue très importante voire vitale. Cette prise de décision doit tenir compte d'une vue globale de l'entreprise. Parmi les moyens qui peuvent donner cette vue globale, on trouve les entrepôts de données qui résultent d'une intégration des données de l'entreprise.

La vue globale à elle seule ne suffit pas pour la prise de décision, car il faut que l'information qu'elle contient soit disponible aux décideurs en temps opportun. L'optimisation du temps d'accès aux informations de l'entrepôt de données est donc primordiale.

Dans le cadre de cette thèse, nous nous sommes intéressé à cet aspect. Plusieurs solutions sont possibles. Nous avons choisi l'approche de la matérialisation des vues. Dans cette approche, il s'agit de precalculer certaines vues et de les matérialiser. Se pose alors le problème de la sélection des vues à matérialiser. Ce problème étant difficile [Gup 99], les méthodes exactes ne sont plus satisfaisantes; nous faisons donc appel aux metaheuristiques pour trouver des solutions proches de l'optimal.

Dans le cadre de cette thèse, nous avons développé une metaheuristique pour résoudre ce problème en mariant deux metaheristiques. Ce mariage à pour intérêt d'atténuer l'aspect aléatoire et d'introduire l'apprentissage dans le fonctionnement de la metaheuristique.

Une étude expérimentale à été menée dans le but d'estimer les performances (rapport de performance et scalabilité) de la méthode.

Mots clés entrepôt de données, vues matérialisée, métaheuristique, cube de données, treillis, coût de maintenance, coût de traitement, optimisation.

Table des matières

Remerciements

Résumé

Introduction …….. 3

1 Les entrepôts de données

1.1 Introduction**…… ……** 6

1.2 Les systèmes décisionnels**……....** 6

1.3 Les Processus OLTP et OLAP**…… ……** 7

1.4 Entrepôt de données**……..** 8

1.5 Modélisation multidimensionnelle……..... 10

1.6 Les implémentations des modèles multidimensionnels…… 12

1.6.1 Les systèmes ROLAP……..... 12

1.6.2 Les systèmes MOLAP…… 12

1.7 Le Cube de données…… 14

1.7.1 Introduction ……. 14

1.7.2 Exploitation du Cube de données ……. 15

1.7.2.1 Roll up (drill down) / Drill down (roll up) …….. 15

1.7.2.2 Slice and Dice …… ….. 16

1.7.2.3 Data surfing …… 16

1.8 Conclusion …….. 16

2 Optimisation du temps de réponse dans un entrepôt de données

PATIE I : Optimisation du temps de réponse par Matérialisation des vues

I.2.1 Introduction…….……. 17

I.2.2 Les vues matérialisées…… 18

I.2.2.1 La sélection des vues à matérialiser.. ……..…. 18

I.2.2.1.1 le problème de la sélection des vues à matérialiser *statique* 20

I.2.2.1.2 le problème de la sélection des vues à matérialiser d*ynamique*. 21

I.2.2.2 Algorithmes de sélection des vues à matérialiser……...…..... 21

I.2.3 Maintenance des vues matérialisées…… 23

I.2.4 Problème de la réécriture des requêtes …… 24

PARTIE II: Quelques notions sure la complexité

II.2.1 Définitions…… 24

II.2.2 Complexité des algorithmes ……..…… 25

II.2.2.1 Calcul de la complexité……..…… 25

II.2.2.2 Classification des algorithmes……. 28

II.2.2.3 Algorithme efficace…… …… 29

II.2.3 Complexité des problèmes……..…….. 29

II.2.4 Classes de complexité…… …… 30

II.2.5 Détermination de la classe d'un problème……...…… 31

II.2.6 Conclusion……..……. 31

3 Les Metaheuristiques pour l'optimisation combinatoire.

3.1 Introduction……...…. 33

3.2 Les méthodes de descente……..…… 34

3.2.1 Schéma général d'une descente…… 35

3.3 Les Algorithmes glouton…… 36

3.4 Le recuit simulé……..… 36

3.5 La recherche tabou ……. 41

3.6 Les algorithmes génétiques…… 43

3.6.1 La sélection ……..…… 44

3.6.2 Le croisement…… …… 45

3.6.3 La mutation…… 47

3.7 Algorithme des Colonies de fourmis…… 48

3.7.1 introduction……..…. 48

3.7.2 Auto organisation……..…… 49

3.7.3 Stigmergie…….…… 51
3.7.4 Contrôle décentralisé…….…… 51
3.7.5Hétérarchie dense…… 51
3.7.6 Principe de fonctionnement des colonies de fourmis …… ….... 53
3.7.7 Les fourmis artificielles…… 55
3.7.8 La métaheuristique ACO (optimisation par colonies de…….. 55
fourmis)
3.7.8.1 Représentation d'un problème par les algorithmes….. 56
ACO
3.7.8.2 Le comportement des fourmis dans l'ACO …… 57
3.7.8.3 Exemple d'adaptation des colonies de fourmis à un 58
problème
3.8 Optimisation par essaim particulaire…… …….. 60
3.9 Conclusion ……...… 62

4 Adaptation des metaheuristiques au problème de la Sélection des vues

4.1 Introduction…….….. 63
4.2 Solution gloutonne ……...… 64
4.2.1 Introduction …… 64
4.2.2 le modèle du coût utilisé ……. 64
4.2.3 l'algorithme glouton (Greedy Algorithme) …… 64
4.3 Solution basée sur la densité (algorithme DVMA)…….. 69
4.3.1 Introduction……. 69
4.3.2 Notion de cluster basé sur la densité…….. 70
4.3.3 Concepts utilisés dans l'algorithme DVMA…… 73
4.3.3.1 le voisinage…… 73
4.3.3.2 la plus proche vue matérialisée parente NMPV …… 74
4.3.3.3 Le bénéfice du voisinage d'une vue ……. 74
4.6.3.4 vue noyau (Core view) …… 75
4.3.3.5 Notion de directement densité-joignable et densité- 75
joignable
4.3.3.6 Les catégories de vues…… 76
4.3.3.7 cluster …….. 76
4.3.4 Le principe de l'algorithme DVMA …… 76

4.4 Conclusion ……... 78

5 Approche hybride développée pour le problème de la sélection des vues.

5.1 Introduction …… 79

5.2 La Modélisation Dimensionnelle …… 80

5.3 Représentation des relations entre les vues matérialisées…… 81

5.4 Formulation du problème…… 83

5.5 Algorithmes génétiques adaptés au problème……. 84

5.5.1 Fonction de pénalité…… 85

5.6 Algorithme de Colonie de fourmis utilisé pour remplacer les opérateurs 86

5.7 Règle de mise à jour de la phéromone…… 88

5.8 Résultats expérimentaux……. 89

5.8.1 Résultats expérimentaux relatifs à l'algorithme proposé….. 92

5.8.2 Comparaisons de l'algorithme proposé avec l'algorithme A* 96

5.8.3 Comparaison de l'algorithme proposé avec l'algorithme EA 97

5.9 Conclusion…… 100

Conclusion générale …….. 101

Bibliographie 103

Liste des figures

1.1 Architecture d'un système décisionnel……. 7

1.2 Architecture d'un entrepôt de données……. 9

1.3 Représentation du fait Vente…… 11

1.4 Représentation des dimensions……. 11

1.5 Schéma en étoile……... 13

1.6 Schéma en flocon…….. 13

1.7 Représentation des données sous forme de cube…… 14

2.1 Schema du processus de la selection des vues à materilisées…… 20

2.2 Arbre algébrique de trois requêtes…… 22

2.3 Les différentes classes de complexité…… 31

3.1 Organigramme général du recuit simulé…… 40

3.2 Organigramme général de la recherche tabou…… 43

3.3 Croisement en un point et en deux points…….. 46

3.4 Organigramme général des algorithmes génétiques……. 47

3.5 : schéma d'Arkin…… 48

3.6 Hiérarchie (a) et hétérarchie dense (b) : deux concepts opposés……. 52

3.7 expérience du nid des fourmis et de la source de nourriture……. 53

3.8 Schéma de l'évitement d'un prédateur par un banc de poissons…… 61

4.1 Treillis de 8 vues……. 65

4.2 Un mauvais exemple pour la solution gloutonne……. 68

4.3 Ensemble témoin de points…… 70

4.4 Point noyau et point frontière……. 71

4.5 Point directement densité-joignable……... 72

4.6 Point densité-joignable……..……. 72

4.7 Point densité-connecté…… 73

4.8 Vue noyau, vue frontière et vue densité joignable .…… 75

5.1 Modèle multidimensionnel représentant le fait vente et trois dimensions .. 81

5.2. Représentation des dépendances entre les vues à l'aide d'un treillis……. 83

5.3 Coût de traitement en fonction du nombre d'individus…… 92

Liste des figures

5.4 Coût de traitement en fonction du paramètre d’évaporation……. 92
5.8 Coût de traitement en fonction de la probabilité d’intensification……. 95
5.9 Faisabilité des solutions en fonction de la probabilité d’intensification…... 95
5.10 Coût de traitement en fonction de la contrainte de maintenance……. 97
5.11 Faisabilité des solutions en fonction de la contrainte de maintenance…… 97
5.12 Coût de traitement en fonction de la contrainte de maintenance…….. 98
5.13 Evolution du temps d'exécution en fonction des dimensions……. 100

Introduction

Toute entreprise, quelque soit sa taille, trouve qu'elle réalise d'importants bénéfices en adoptant un entrepôt de données. Il est généralement admis que les entrepôts de données fournissent une excellente approche pour transformer les grands volumes d'information qui existent dans ces entreprises en une information utile pour supporter le processus de prise de décision. Un entrepôt de données fournit une base pour les techniques d'analyse telles que la fouille de données et l'analyse multidimensionnelle.

Un des facteurs importants dans un entrepôt de données est le temps de réponse aux requêtes des décideurs. Le volume important des entrepôts de données implique une augmentation de ce temps. Apparaît alors la nécessité de techniques pour réduire ce temps. Une des techniques utilisées est la matérialisation des vues. Lorsqu'une requête est introduite, on tentera d'y répondre en utilisant ces vues plutôt que d'accéder à l'entrepôt de données.

Se pose alors le problème du choix de l'ensemble de vues à matérialiser pour minimiser le temps de réponse global tout en respectant certaines contraintes liées aux ressources. Les algorithmes exacts de la recherche opérationnelle ne donnent des résultats satisfaisants que pour des tailles réduites du problème. Pour des tailles plus importantes, des heuristiques sont utilisées pour donner des solutions proches de l'optimal.

Dans l'environnement d'un entrepôt de données, il est généralement possible d'isoler un ensemble de requêtes à privilégier. L'ensemble des vues à matérialiser doit être déterminé en fonction de cet ensemble de requêtes.
Le problème des vues matérialisées peut être vu de deux manières différentes suivant le modèle utilisé.
Dans le cas du modèle MOLAP, on utilisera une structure de treillis (hypercube) où chaque nœud représente une vue. Dans le cas du modèle ROLAP, chaque requête est

représentée par un arbre algébrique. Chaque nœud non feuille représente une vue potentielle

Dans le cadre de cette thèse nous nous plaçons dans le cadre du modèle MOLAP pour étudier l'optimisation du temps de réponse des requêtes. Nous proposons une nouvelle approche qui se base sur une hybridation des algorithmes génétiques avec les colonies de fourmis. Cette hybridation a pour effet d'atténuer l'aspect aveugle des opérateurs génétiques et d'introduire l'aspect apprentissage grâce à la phéromone de la fourmi.

Cette thèse est organisée comme suit :
Dans le chapitre 1, nous donnons quelques notions sur des entrepôts de données. Nous présentons notamment, les systèmes décisionnels, la modélisation dimensionnelle, ainsi que les implémentations possibles des modèles multidimensionnels.

Dans le chapitre 2, nous passons sommairement en revue quelques travaux faits pour résoudre le problème de la sélection des vues à matérialiser dans un entrepôt de données. Nous presentons une solution sans contraite, une solution avec la contrainte espace et une solution avec la contraite coût de maintenance. Du fait que le problème traité est NP-Difficile, nous introduisons en fin du chapitre quelques notions de complexité.

Le chapitre 3 sera consacré aux métaheuristiques les plus connues. Nous traiterons des metaheuristiques les plus connues, à savoir, les algorithmes glouton, le recuit simulé, la recherche tabou, les algorithmes génétiques, les colonies de fourmis et les algorithmes d'optimisation par essaim particulaire.

La résolution du problème de la matérialisation des vues sera étudiée au chapitre 4. Nous y présentons deux solutions :

- La première solution est une solution gloutonne, elle considère la contrainte espace de stockage et utilise la notion de bénéfice pour construire la solution.

- La deuxième solution est basée sur la notion de densité et cluster, elle utilise la notion du bénéfice du voisinage et fonctionne sans contrainte.

Le dernier chapitre (chapitre 5) est consacré à l'approche hybride que nous avons développée pour traiter du problème de la selection des vues matérialisées. Nous y décrivons l'approche à travers la formulation du problème, la representation des relations de dependance entre les vues et la description du principe de l'hybridation proposée. Des resultats expérimentaux relatifs à notra travail sont présentés pour valider l'approche proposée.

Chapitre 1

Les entrepôts de données

Objectif : *l'objectif de ce chapitre est de donner une vue d'ensemble sur les entrepôts de données. On y définira les systèmes décisionnels, les entrepôts de données ainsi que la notion de cube de données et de modélisation dans les entrepôts de données.*

1.1 Introduction

L'entreprise d'aujourd'hui doit être capable d'anticiper sur les changements, les nouveaux besoins des clients et d'anticiper par rapport à la concurrence. Cela implique que son système d'information fonctionne avec pertinence en temps voulu. Dans le processus de prise de décision, l'intuition est importante mais insuffisante, il faut aussi pouvoir prendre la décision informée. Constituer ce gisement d'information pertinente et organiser sa distribution devient le rôle des systèmes décisionnels et de l'entrepôt de données. **[Fra 01]**

1.2 Les systèmes décisionnels

Définition 1.2.1

Un système décisionnel est un système d'information dédié aux applications décisionnelles, il regroupe un ensemble d'informations et d'outils mis à la disposition des décideurs pour supporter de manière efficace la prise de décision **[Tes 00].**
Ces systèmes mettent en jeux quatre éléments essentiels. (figure 1.1) :

- **Les sources de données :** les sources de données sont nombreuses, variées, distribuées et autonomes. Elles peuvent être internes (base de production) ou externes (base de partenaires) à l'entreprise.
- **L'entrepôt de données :** l'entrepôt de données est le lieu de stockage centralisé des informations utiles pour les décideurs.
- **Les magasins de données :** les magasins de données sont des extraits de l'entrepôt orientés sujet. Les données sont organisées de manière adéquate pour permettre des analyses rapides à des fins de prise de décision.
- **Les outils d'analyse :** les outils d'analyses permettent de manipuler les données suivant des axes d'analyses. L'information est visualisée à travers des interfaces interactives et fonctionnelles dédiées à des décideurs souvent non informaticiens (directeur, chef de service,…)

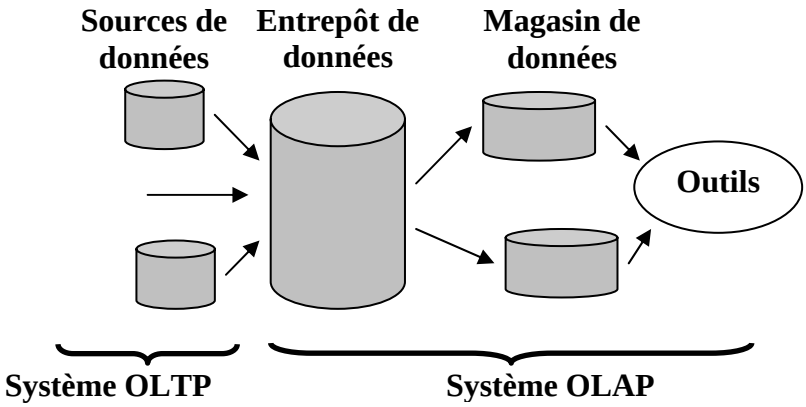


Figure 1.1 Architecture d'un système décisionnel

1.3 Les Processus OLTP et OLAP

Les modes de travail OLAP et OLTP sont totalement différents.

OLTP (On line transactional processing) sont centrés sur la mise a jour ponctuelle des données, ils travaillent sur les versions les plus récentes des données. Ils se caractérisent par ce qui suit :

- Ils sont nombreux au niveau d'une entreprise.
- Ils s'intéressent surtout à la mise à jour des données.
- Ils traitent un volume réduit d'information
- Ils sont exécutés par de nombreux utilisateurs

OLAP (On Line Analytical Processing) par contre travaillent en lecture seulement, et consultent d'importantes bases de données dans le but d'analyses. Ils se caractérisent par ce qui suit :

- Ils sont peu nombreux au niveau de l'entreprise
- Ils visent à interroger visualiser et synthétiser les données
- Ils concernent un nombre important d'enregistrements
- Ils sont utilisés par un nombre réduit de décideurs **[Tes 00]**

Les Solutions OLAP ont pour but de restructurer et de stocker dans un format multidimensionnel les données provenant des bases de données opérationnelles. Dans ce format multidimensionnel, les données sont organisées le long de dimensions. Ce qui permet des analyses suivant divers axes.

1.4 Entrepôt de données

Il existe plusieurs définitions d'un entrepôt de données. On pourrait le définir comme étant une collection de données **intégrées**, **orientées sujet**, **non volatiles**, **historisées**, **résumées**, et **disponible pour l'interrogation et l'analyse**.

- ***Données intégrées***: l'entrepôt de données puise ces données de différentes sources qui souvent, sont hétérogènes. L'intégration résout les problèmes d'hétérogénéité des systèmes de stockage, des modèles de données et de sémantique des données.
- ***Orientées sujet***: Dans l'entrepôt, les données sont réorganisées autour de thèmes tels que la vente, stock …etc. L'intérêt de cette organisation est qu'il devient possible de réaliser des analyses qui serviront à la prise de décision.
- ***Non volatiles***: On ne doit pas supprimer les données introduites dans l'entrepôt de données pour assurer la traçabilité des informations et des décisions prises. Les données de l'entrepôt sont donc utilisées essentiellement en mode consultation.

- ***Données historisées:*** l'évolution des données est essentielle pour la prise de décision. Elle doit être prise en compte. Elle peut servir dans la prédiction des évolutions futures en fonction des évolutions passées. Un référentiel temps doit être associé aux données.
- ***Résumés:*** Les informations issues des sources de données doivent être agrégées et réorganisées afin de faciliter le processus de prise de décision.
- ***Disponible pour l'interrogation et l'analyse***: Des droits d'accès doivent être définis pour les utilisateurs de l'entrepôt de données. Des outils interactifs doivent permettre la manipulation et l'analyse des données.

Un entrepôt de données doit permettre l'accès aux informations de l'entreprise, ces informations doivent être cohérentes, et de qualité.
Il existe plusieurs architectures d'entrepôt de données. La figure 1.2 présente l'une des architectures possibles.

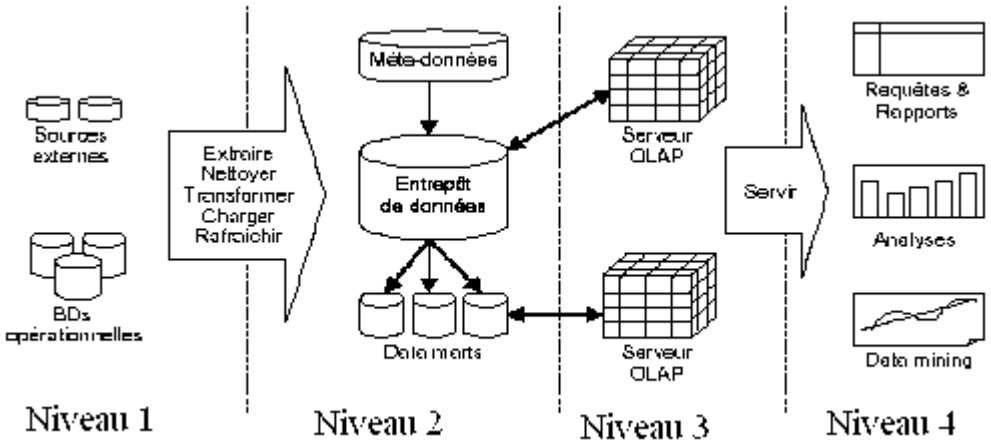


Figure 1.2. Architecture d'un entrepôt de données.

Dans cette architecture

- le premier niveau est occupé par les sources de données. Elles sont constituées des systèmes opérationnels d'enregistrement, dont la fonction consiste à capturer les transactions liées à l'activité de l'entreprise. Ce sont les bases de production.

- Le second Niveau est occupé par l'entrepôt proprement dit qui est alimenté par les sources de données. On trouve aussi dans ce niveau, les meta-données qui sont des informations sur l'environnement de l'entrepôt de données. Elles regroupent l'ensemble des informations concernant l'entrepôt de données. Les magasins de données sont la troisième composante du second niveau, ce sont des extraits de l'entrepôt. Les données extraites sont adaptées à une classe de décideurs ou à un usage particulier.
- Le niveau 3 est constitué des serveurs de présentation qui répondent aux requêtes des décideurs, des générateurs d'état ou d'autres applications.
- Le dernier niveau est le portail de restitution, il représente ce que voient les utilisateurs, il regroupe toutes les applications qui s'appuient sur les données de l'entrepôt pour les restituer soit à l'utilisateur soit a une application.

1.5 Modélisation multidimensionnelle

Le modèle entité-association est souvent utilisé pour la conception des bases de données. Ce modèle permet d'éliminer les redondances en introduisant un grand nombre de nouvelles entités. De ce fait l'accès aux données devient plus compliqué et le schéma difficile à lire. L'utilisation de ce modèle pour la conception des entrepôts de données n'est donc pas appropriée **[Kim 97].**

La modélisation multidimensionnelle est une méthode de conception associée aux entrepôts de données. Elle consiste à considérer un sujet analysé comme un point dans un espace multidimensionnel. Le constructeur fondamental des modèles multidimensionnels est le cube de données.

Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives d'analyse **[Tes 00]**.

Cette modélisation a donné naissance aux concepts de fait et de dimension **[Kim 96]**

Le fait représente le sujet analysé, il est formé de mesures correspondantes aux informations de l'activité analysée.

Exemple : Si nous prenons comme sujet d'analyse les ventes et que nous nous intéressons à la quantité vendue et au montant global des ventes, nous allons

concevoir le fait Vente dont les mesures sont Quantité et Montant. Ce fait sera représenté par un rectangle de la manière suivante (figure 1.3) :

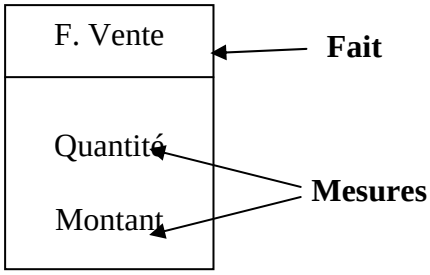


Figure 1.3. Représentation du fait Vente

Une dimension représente une perspective d'analyse. Elle est formée de paramètres faisant varier les mesures de l'activité. Les paramètres sont hiérarchisés du plus faible niveau au plus haut niveau. Cette hiérarchisation sert à diminuer ou augmenter les niveaux de détail de l'analyse.

Exemple :

Le fait vente de l'exemple précèdent peut être analysé suivant diverses dimensions. Nous pouvons prendre la dimension temps, la dimension lieu et la dimension client. Ces dimensions seront représentées de la manière suivante (figure 1.4) .

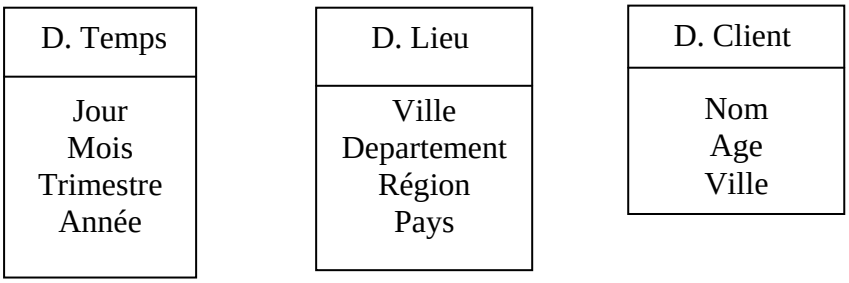


Figure 1.4 Représentation des dimensions

Dans la dimension Lieu, nous pouvons définir la hiérarchie suivante :

Ville ⟶ Département ⟶ Région

Où Ville est le paramètre le plus détaillé (le plus fin) de la hiérarchie.

1.6 Les implémentations des modèles multidimensionnels

Il existe deux principales approches pour implémenter le cube de données. Les deux approches différent dans la manière de stocker le cube de données.

1.6.1 Les systèmes ROLAP

Ce type de systèmes utilise un SGBD relationnel pour stocker les données. Il est conçu comme une couche supplémentaire au dessus du SGBD et fournit une vision multidimensionnelle des données. Les mesures sont stockées dans une table appelée *table des faits***,** les dimensions sont stockées dans des tables de dimensions**.**

Ces systèmes peuvent supporter de grands volumes de données mais le temps de réponse peut être élevé.

1.6.2 Les systèmes MOLAP

Dans ce type de système les données sont stockées dans des tableaux multidimensionnels. Chaque dimension du cube est associée à une dimension du tableau. La valeur de la cellule dans le cube est stockée dans le tableau. Ces systèmes sont performants mais difficile à mettre à jour et à gérer. **[Vas 99]**

Ils sont bien adapté à des entrepôts dont la taille ne dépasse pas quelques gigaoctets et dont le modèle n'évolue pas beaucoup. **[Gue 01]**

Trois Schémas sont utilisés pour modéliser les systèmes multidimensionnels:

- Le schéma en étoile,
- le schéma en flocon de neige et
- le schéma en constellation.

Le schéma en étoile est constitué du fait central et des dimensions qui représentent visuellement une étoile (figure 1.5)

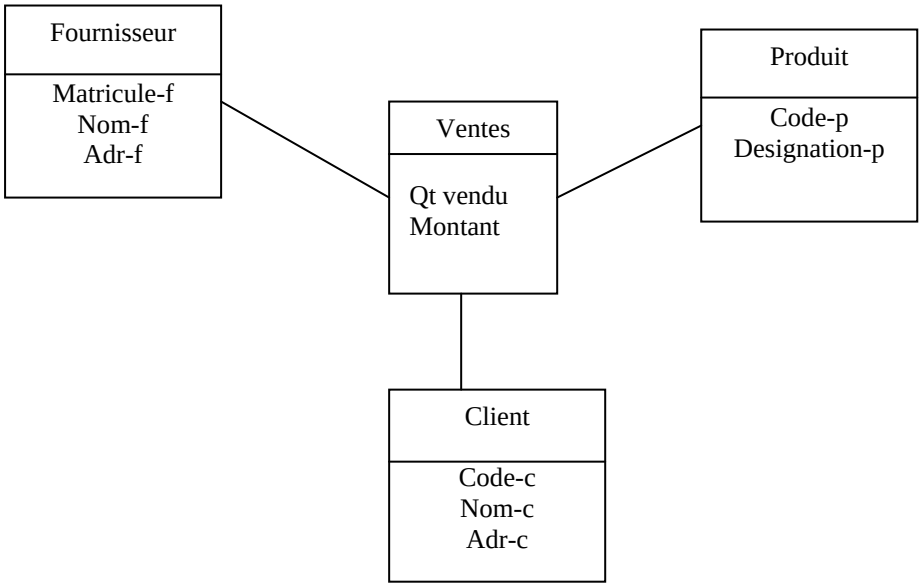


Figure 1.5 Schéma en étoile

Dans le schéma en flocon, les dimensions du modèle en étoile sont décomposées conformément à la hiérarchie des paramètres.(figure 1.6)

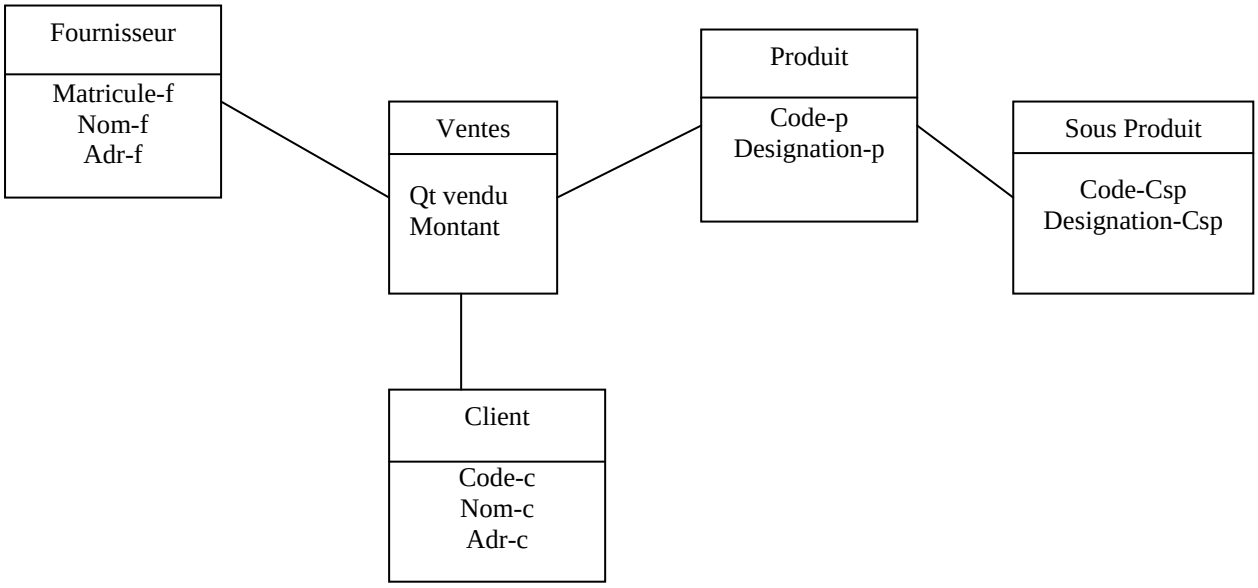


Figure 1.6 Schéma en flocon

Dans le schéma en constellation, plusieurs modèles en étoile sont fusionnés en mettant éventuellement des faits et des dimensions en commun.

1.7 Le Cube de données

1.7.1 Introduction

Les serveurs OLAP ont été conçus pour s'intégrer dans un environnement client/serveur afin d'en retirer les possibilités offertes. Les décideurs disposant de postes de travail intelligents accèdent à un serveur de base de données multidimensionnelle. Celui-ci contient un hypercube (cube à n dimensions) prédéfini dans lequel doit être stockée la globalité des données. Ce qui nécessite de s'appuyer sur une information pré-packagée et fortement structurée. Il permettra ainsi d'analyser la répartition d'un indicateur comme le " *chiffre d'affaire*" en fonction des axes ou dimensions " *clients* ", " *produit* ", " *temps*". En outre, des hiérarchies seront définies pour chaque axe d'analyse. Une fois cette structure multidimensionnelle établie, l'outil OLAP propose des méthodes de navigation dans les données, comme le "drill-down" pour aller vers les informations détaillées dans une hiérarchie, le "slice and dice" pour changer d'axe d'analyse

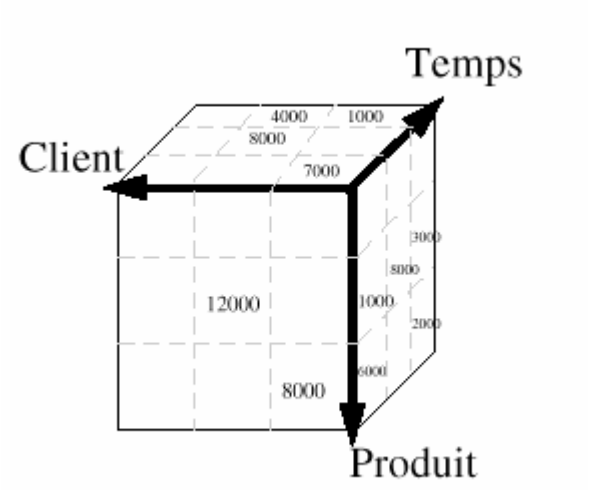


Figure 1.7 - Représentation des données sous forme de cube.

1.7.2 Définition

Le cube de données est une Structure multidimensionnelle permettant l'analyse d'informations factuelles en les segmentant sur un ensemble d'axes d'analyse, un axe d'analyse représente une dimension.

Le cube de données est exploré afin de trouver les informations les plus pertinentes, la valeur de chaque cellule dans le cube est une mesure qui correspond aux différentes dimensions. Comme exemple on considère un entrepôt de données qui a trois dimensions : produits, fournisseurs et clients. La mesure qui nous intéresse est la vente totale. Donc pour chaque cellule (p, f, c) dans ce cube de données, on enregistre les ventes totales d'un produit p, acheté d'un fournisseur f, et vendu à un client c

1.7.3 Exploitation du Cube de données

L'intérêt du cube de données est la possibilité de réaliser facilement des coupes par sélection selon une dimension, et des projections avec agrégations des mesures par groupement selon la dimension éliminée. Ces opérations sont visuellement faciles à spécifier. Elles permettent des analyses plus au moins fines des données. Elles correspondent à la dérivation de vues concrètes avec agrégats **[Gar00]**.

1.7.3.1 Roll up (drill down) / Drill down (roll up)

Roll up désigne la faculté d'aller du niveau global vers le niveau détaillé, et Drill down est l'inverse. Ce mécanisme est totalement basé sur la notion d'hiérarchie. Chacun des axes d'analyse se décompose en attributs reliés entre eux par des relations père /fils.

comme il a été cité précédemment une dimension doit normalement pouvoir comporter plusieurs hiérarchies. Par exemple, la dimension " *produits* " peut contenir une hiérarchie " *Marque-Article* " et une hiérarchie " *Secteur-Segment-Article* ". Le mécanisme de drill-down se fera ainsi de la marque vers l'article et du secteur vers le segment puis vers l'article. La mise en œuvre de cette fonctionnalité n'est cependant pas toujours aussi simple. En effet, une dimension peut contenir des hiérarchies partant d'un même attribut pour aller vers un attribut différent. Par exemple, l'axe " *Clients* " peut contenir la hiérarchie " *Client-Individu* " et la hiérarchie " *Client-Société-Filiale* ". L'outil doit alors permettre soit de redescendre de Client vers à la fois Individu et Société puis de redescendre au niveau Filiale pour les clients de type Société, soit plus simplement, permettre à l'utilisateur de choisir de redescendre vers le niveau Individu ou vers le niveau Société.

1.7.3.2 Slice and Dice

Le " *Slice and Dice* " désigne la possibilité de faire pivoter dynamiquement les axes du tableau d'analyse croisé. Il est possible ainsi de passer d'un tableau présentant les ventes par magasin en lignes et jour en colonnes par un tableau similaire présentant les ventes par magasin en colonnes et jour en lignes.

1.7.3.3 Data surfing

Le data surfing est la possibilité laissée à l'utilisateur de circuler librement, de manière intuitive et ergonomique dans un modèle dimensionnel, au delà d'un " simple " drill-down ou slice and dice. L'utilisateur peut alors modifier dynamiquement ses axes d'analyse ou appliquer un nouveau filtre à ses données. Ces mécanismes s'appliquent sur le modèle défini soit par l'administrateur, soit par l'utilisateur.

Ce modèle doit être assez complexe pour adresser l'ensemble des demandes de l'utilisateur et assez souple pour que l'utilisateur puisse le personnaliser à son goût, en créant de nouveaux attributs ou de nouveaux axes d'analyse ou en définissant de nouvelles métriques calculées.

1.8 Conclusion

Nous venons de voir dans ce chapitre quelques notions sur les entrepôts de données qui sont devenus une nécessité pour la prise de décision dans une entreprise. Cette prise de décision doit être faite en temps opportun, d'où la nécessité d'avoir des temps de réponse réduits. Le volume important des entrepôts de données fait que ce temps de réponse croit continûment. La matérialisation des vues est l'une des techniques utilisées pour réduire ce temps. Se pose alors la question : Quelles vues matérialiser ? Il a été montré que ce problème est difficile [Gup99]. Dans le chapitre suivant, nous allons parler de l'optimisation du temps de réponse par matérialisation des vues et de la complexité des problèmes et des algorithmes. Nous allons citer sommairement quelques travaux qui résolvent le problème

Chapitre 2

Optimisation du temps de réponse dans un entrepôt de données

Objectif : *Ce chapitre est divisé en deux parties. Dans la première partie, nous parlons de l'optimisation des requêtes par matérialisation des vues et nous citons de manière sommaire quelques travaux dans ce domaine. Etant donné que le problème traité est NP-difficile, nous parlons dans la deuxième partie de quelques notions de complexité.*

PATIE I : Optimisation du temps de réponse par Matérialisation des vues

I.2.1 Introduction

Hormis l'analyse et la prise de décision, l'un des objectifs des entrepôts de données est de fournir un accès permanent aux données même lorsque les bases de données individuelles sont inaccessibles, et de réduire les accès distants au système gérant les données.

Le processus d'analyse est réalisé à l'aide de requêtes complexes comportant de multiples jointures et des opérations d'agrégation sur un volume important qui ne cesse d'augmenter. De ce fait le temps de réponse se dégrade de plus en plus.

L'administrateur, dans le but de minimiser le coût d'exécution des requêtes, sélectionne un ensemble de vues à matérialiser et un ensemble d'index. Cette sélection permet de diminuer le coût des requêtes mais entraîne deux problèmes

- Augmentation de l'espace de stockage
- Augmentation du coût de maintenance

Ce sont les industriels qui se sont d'abord intéressés aux entrepôts de données, mais par la suite les chercheurs se sont penchés sur ce concept. Ces deux communautés ont collaboré pour atteindre un niveau de performance acceptable. Elles ont développé des techniques d'optimisation du temps de réponse. Elles se sont surtout intéressées à la phase de conception logique de l'entrepôt (sélection et maintenance des vues matérialisées) et à la phase conception physique (sélection des index).

I.2.2 Les vues matérialisées

Une vue peut être désignée par l'expression de la requête qui la définit, une vue matérialisée est une table contenant le résultat d'une requête. La vue matérialisée a donc une existence physique. La matérialisation des vues améliore l'exécution des requêtes en précalculant les opérations les plus coûteuses ou des ensembles d'enregistrements qui évitent l'accès à l'entrepôt de données. Donc certaines requêtes ne nécessitent l'accès qu'aux vues matérialisées.

Dans le contexte OLTP, les vues sont utilisées pour satisfaire d'autres objectifs tels que la sécurité, la confidentialité, etc.…

La matérialisation des vues permet d'améliorer les performances des requêtes, mais elle peut être aussi utilisée pour fournir des données dupliquées.

Les deux grands problèmes de la matérialisation des vues sont

1. le problème de la sélection des vues à matérialiser
2. le problème de la maintenance des vues matérialisées

I.2.2.1 La sélection des vues à matérialiser

Dans un entrepôt de données, il est possible de spécifier un ensemble de requêtes à privilégier. En fonction de ces requêtes, l'ensemble des vues à matérialiser sera défini.

Selon le type du modèle de données utilisé (MOLAP ou ROLAP), le problème de la sélection des vues à matérialiser sera vu différemment.

- Dans le cas du modèle du type MOLAP, le cube de données est considéré comme structure principale. Chaque cellule du cube est considérée comme

vue potentielle. Dans ce cas une structure de treillis sera utilisée pour déterminer les vues à matérialiser.

- Dans le cas du modèle ROLAP, chaque requête est représentée par un arbre algébrique, chaque nœud non feuille est considéré comme vue potentielle

Dans le problème de la sélection des vues à matérialiser, il existe trois possibilités

1. matérialiser toutes les vues : Cette approche donne le meilleur temps de réponse, mais induit un coût de stockage et de maintenance important
2. ne matérialiser aucune vue : cette approche ne coûte rien en terme de stockage et de maintenance. Mais n'apporte aucun avantage en terme de temps de réponse
3. Dans le mode MOLAP, il existe une certaine dépendance entre les requêtes. C'est-à-dire que certaines requêtes peuvent être définies à partir d'autres. La sélection des vues doit tenir compte de cette dépendance pour définir les vues à matérialiser.
   Dans le modèle ROLAP, on trouve certains nœuds communs. Il est alors souhaitable de matérialiser les parties partagées.

Le problème de la sélection des vues à matérialiser peut être vu de la manière suivante. **[Gup 99]:**
Etant donnée une contrainte de ressource S (Capacité de stockage ou coût de maintenance), le problème de la sélection des vues à matérialiser consiste à sélectionner un ensemble de vues $\{V_1,V_2,\ldots,V_n\}$ minimisant une fonction objectif (coût de traitement des requêtes) et satisfaisant la contrainte (figure2.1).

Les méthodes exactes qui résolvent se problème tentent d'énumérer toutes les vues possibles, afin de sélectionner un ensemble de vues optimales. Leur nombre est en général très grand. Si d est le nombre de dimensions dans un schéma, alors le nombre d'agrégations possibles est égal à $n=2^d$ et la complexité est de $O(2^n)$.
Il a été montré que ce problème est NP-difficile [Gup 99].

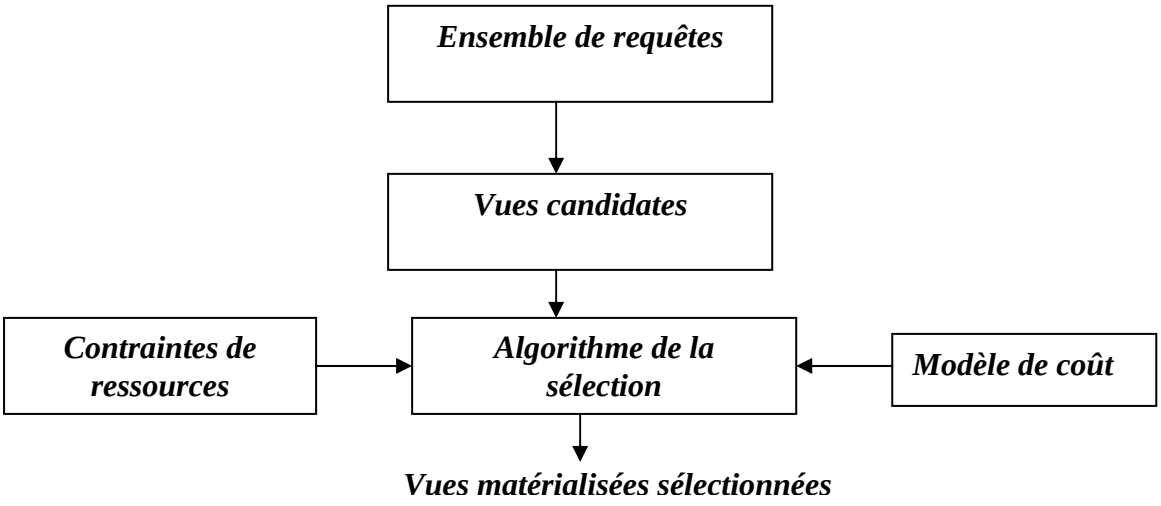


Figure 2.1 Schéma du processus de la séléction des vues à materiliser

La matérialisation des vues induit un autre problème qui est la connaissance préalable ou non de l'ensemble des requêtes. D'où la naissance de deux catégories de problèmes.

1. le problème de la sélection des vues à matérialiser *statique.*
2. le problème de la sélection des vues à matérialiser *dynamique.*

I.2.2.1.1 le problème de la sélection des vues à matérialiser *statique*

Dans ce problème, nous avons trois données:

- Le schéma de l'entrepôt
- L'ensemble des requêtes les plus fréquemment utilisées
- Une contrainte de ressource

Dans ce cas le problème consiste à sélectionner un ensemble de vues à matérialiser afin de minimiser le coût total d'évaluation des requêtes sous la contrainte de la ressource.

Il est supposé ici que l'ensemble des requêtes n'évolue pas. Dans le cas d'une évolution, il est nécessaire de reconstruire l'ensemble des vues.

Plusieurs solutions ont été proposées à ce problème, en prenant en compte la contrainte de la ressource de manière différente. Nous rappelons que la contrainte peut être l'espace, le coût de maintenance ou bien les deux.

La contrainte espace de stockage a été traitée dans **[Shu 98]**, **[Har 96]** et **[Gup 97]**
La contrainte du coût de maintenance a été traitée dans **[Gup 99]**,**[Cho 02]** et **[Jef 03]**
[Zha 01] propose un algorithme génétique pour ce problème sans aucune contrainte. **[Lee 01]** propose un algorithme génétique pour ce problème sous la contrainte coût de maintenance en introduisant une fonction de pénalité.
[Jef 03] présente un nouvel algorithme génétique sans utiliser de fonction de pénalité. Il introduit une procédure de rangement stochastique.

I.2.2.1.2 le problème de la sélection des vues à matérialiser

Dynamique

Afin d'éviter de reconstruire entièrement les vues matérialisées en cas d'évolution des requêtes, Kotidis et al. [KoR 86] on proposé un système qui matérialise les vues de manière dynamique. Ce système enregistre les évolutions des requêtes matérialisées et matérialise à chaque fois le meilleur ensemble de vues pour satisfaire ces requêtes. Il rafraîchit pendant les opérations de mise à jour, les vues et si la contrainte espace, n'est pas satisfaite, il élimine les vues les moins utilisées.

De nombreux algorithmes ont été proposés pour élaborer une solution optimale ou proche de l'optimale. La plupart de ces algorithmes sont destinés au problème statique.
Nous allons passer en revue de manière sommaire quelques travaux.

I.2.2.2 Algorithmes de sélection des vues à matérialiser

Nous pouvons classer les algorithmes pour la sélection des vues à matérialiser en trois catégories selon le type de la contrainte utilisée.

- *Les algorithmes sans aucune contrainte*
- *Les algorithmes dirigés par la contrainte espace*
- *Les algorithmes dirigé par la contrainte du temps total de maintenance des vues.*

A) Algorithme sans contraintes (travaux de Yang et al. [YKL 97]).

Les auteurs ont développé un algorithme de sélection des vues à matérialiser dans un contexte ROLAP statique. Ils partent du principe que les requêtes décisionnelles partagent certaines expressions. Leur algorithme procède de la manière suivante :

Chaque requête est représentée par un arbre algébrique. L'algorithme essaye de trouver les expressions communes entre ces arbres (ou nœuds partagés). Les arbres sont ensuite fusionnés en un seul graphe appelé *plan multiple d'exécution des vues* en utilisant les nœuds partagés. Ce graphe a plusieurs niveaux :

- Dans le niveau 0, nous trouvons les tables de base de l'entrepôt.
- Dans le niveau 1, nous trouvons les nœuds qui représentent les opérations ensemblistes tels que l'union, la jointure, etc.
- Dans le niveau 2, nous trouvons le résultat de la requête.

Chaque nœud du niveau 2 est pondéré par le coût de l'opération algébrique et le coût de maintenance. L'algorithme utilise ce graphe pour rechercher l'ensemble des vues à matérialiser pour minimiser le coût de traitement. Chaque nœud du niveau 2 est considéré comme une vue potentielle.

Exemple : considérons un schéma d'entrepôt ayant 05 tables. On définit trois requêtes sur ces tables. Nous remarquons que les requêtes Q1 et Q2 ont une expression commune. Ces deux nœuds sont de bons candidats à la matérialisation

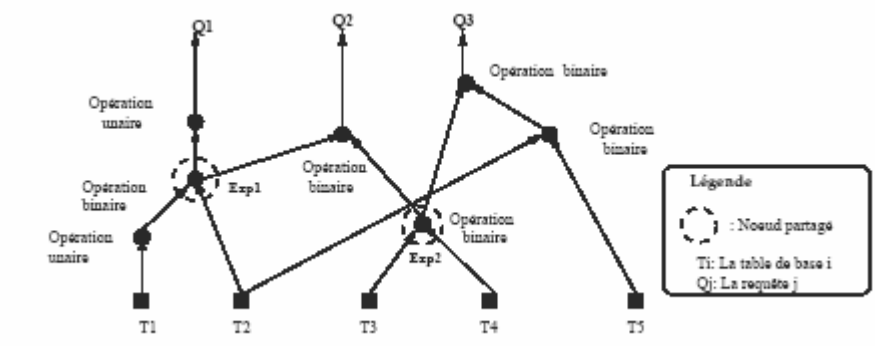


Figure 2.2 Arbre algébrique de trois requêtes

B) Algorithme dirigé par la contrainte espace (travaux Harinarayan et al. [Har 96]).

Dans ce travail les auteurs ont présenté un algorithme glouton pour sélectionner un ensemble de vues à matérialiser. L'objectif de cet algorithme est de minimiser le temps de réponse des requêtes sous la contrainte espace de stockage. Le détail de cet algorithme sera vu au chapitre 4.

C) Algorithme dirigé par le temps de maintenance (travaux de Jeffrey et al.[jef 03]

Les auteurs utilisent un algorithme évolutionnaire. Les dépendances entre les vues sont représentées par une structure de treillis. Ils utilisent un modèle de coût linéaire. Ils minimisent le temps de traitement global sous la contrainte du coût de maintenance.

I.2.3 Maintenance des vues matérialisées

Un entrepôt de données est le résultat d'une intégration de plusieurs sources de données, et contient un ensemble de vues matérialisées. Les sources de données évoluent à cause des mises à jour. Cependant, Si ces changements ne sont pas répercutés sur les vues matérialisées, leurs contenus deviendront obsolètes et ne représenteront plus la réalité. Afin de résoudre ce problème d'inconsistance des données, une procédure de maintenance des vues doit être mise en place. Cette procédure peut être réalisée de trois manières différentes :

- Mise à jour périodique des vues. Les vues sont considérées comme des photographies à des instants définis **[Lin 86]**
- Mise à jour immédiate à la fin de chaque transaction.**[Bla 86]**
- Propagation de manière différée de la mise à jour. Dans ce cas la vue est mise à jour au moment où elle est utilisée

Pour ces trois stratégies, la maintenance ne signifie pas de recalculer toutes les vues car cette approche est très coûteuse. Une bonne maintenance des vues est réalisée

lorsque les changements effectués sur les sources peuvent être propagés aux vues sans recalculer complètement leur contenu.

I.2.4 Problème de la réécriture des requêtes

Les vues matérialisées sont stockées physiquement. Cela permet de les utiliser pour répondre aux requêtes. Après la sélection des vues à matérialiser toutes les requêtes définies sur l'entrepôt doivent être réécrites en fonction de ces vues. Ce processus est appelé *réécriture des requêtes*. Il a été utilisé pour optimiser le coût d'évaluation des requêtes.

Plus formellement : Soit une requête Q définie sur un schéma d'une base de données et un ensemble de vues $\{V_1,V_2 ,\ldots,V_n\}$, sur le même schéma. Est il possible de répondre à la requête Q en utilisant uniquement ces vues.

La sélection des vues à matérialiser est l'une des techniques pour optimiser le temps de réponse dans un entrepôt de données. D'autres techniques sont utilisées comme la fragmentation et la sélection des index. Dans le cadre de cette thèse nous nous sommes intéressés à la matérialisation des vues. Nous avons déjà dit que ce problème est NP difficile. Dans ce qui suit nous allons parler de complexité des algorithmes et de complexité des problèmes.

PARTIE II: Quelques notions sur la complexité

II.2.1 Définitions

Définition 2.1.1 *Un problème est une question à laquelle on veut apporter une réponse. Tout problème comporte dans sa description des paramètres formels. En donnant des valeurs à ces paramètres on obtient une instance du problème. Résoudre un problème c'est donner une solution à toutes ses instances possibles.*

Définition 2.1.2 *Un algorithme de résolution d'un problème P donné est une procédure, décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de n'importe quel exemple du problème P en une chaîne de caractères représentant les résultats de P.*

Le mot algorithme provient du nom du Mathématicien du IXe siècle, Mohammed ibn-Moussa al-Khawarizmi. Autant dire que les algorithmes, au sens de solution à des problèmes, sont connus et utilisés bien avant les débuts de l'informatique. Il arrive souvent que des algorithmes conçus pour résoudre le même problème différent fortement entre eux. Ces différences peuvent être bien plus importantes que celles dues au matériel, aux logiciels ou encore aux programmeurs. Il faut donc trouver un moyen pour juger l'efficacité d'un algorithme et pour comparer plusieurs algorithmes entre eux sans tenir compte des moyens utilisés.

Définition 2.1.3 La *complexité des algorithmes est l 'étude de l'efficacité des algorithmes. Cette complexité mesure le temps ainsi que l'espace nécessaire à un algorithme pour résoudre un problème.*

II.2.2 Complexité des algorithmes

Dans ce qui suit on s'intéresse à la complexité temporelle

Définition 2.2.1 *Le temps d'exécution d'un algorithme est le nombre d'opérations élémentaires (accès à une cellule mémoire, comparaison de valeurs, opérations arithmétiques,...) qu'il effectue. On dit que la complexité de l'algorithme est O(f(n)) où f est d'habitude une combinaison de polynômes, logarithmes ou exponentielles quand le nombre d'opérations effectuées est borné par cf(n), où c est une constante positive, lorsque n tend vers l'infini (n est la taille du problème).*

II.2.2.1 Calcul de la complexité

Voici quelques exemples pour illustrer le calcul de la complexité.

Idée 1: évaluer l'algorithme sur des données de grande taille, lorsque n est grand,

$2n^3 + 5n^2$ est essentiellement $2n^3$.

Idée 2: on élimine la constante multiplicatrice de 2 x n^3, on ne retient que n^3.
L'algorithme est dit en $O(n^3)$.

Exemple 1 Calcul de la somme des éléments d'un vecteur V à n éléments.

```
S := 0;
Pour i := 1 à n
faire
    S := S + V [i];
Fait;
```

Le temps d'exécution de cet algorithme est égal à C1 + n(C2 + C3) où :
C1 est le temps d'exécution de l'affectation S := 0,
C2 est le temps d'exécution de l'incrémentation et du test de la variable i dans la boucle,
C3 est le temps d'exécution de l'instruction S := S + V [i] (affectation et somme).
C1, C2 et C3 sont des constantes, donc le temps d'exécution de l'algorithme s'écrit an+b = O(n) qui est une fonction linéaire en n, où n est le nombre d'éléments du vecteur V ou la taille de l'entrée. On dit que l'algorithme s'exécute en O(n) ou que la complexité de l'algorithme est en O(n).

Exemple 2: Recherche d'une valeur val donnée dans un vecteur V à n éléments

```
i := 1;
Tant que (i ≤ n) et (V [i] ≠val)
faire
    i := i + 1;
Fait;
trouve := i ≤ n;
```

Le temps d'exécution de cet algorithme est égal à C1 + nb (C2 + C3) + C4
C1 est le temps d'exécution de l'affectation i := 1,
nb est le nombre de fois que la boucle est exécutée,
C2 est le temps d'exécution du test de la boucle,

C3 est le temps d'exécution de l'instruction i := i + 1,
C4 est le temps d'exécution de l'instruction trouve := i ≤ n.
C1, C2, C3 et C4 sont des constantes, donc le temps d'exécution de l'algorithme s'écrit
(a * nb + b). On remarque que ce temps d'exécution dépend des éléments du vecteur. Dans le cas le plus favorable (nb = 0, la valeur val est à la première position),
le temps d'exécution est une constante et est noté O(1). Dans le cas le plus défavorable (nb = n, la valeur val n'existe pas), le temps d'exécution est égal à a*n + b = O(n). On dit dans ce cas que l'algorithme s'exécute en O(n).

Exemple 3 Calcul du déterminant par la méthode directe de Cramer
Un déterminant d'ordre 2 se calcule de la manière suivante :

$$\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} = a_{00}a_{11} - a_{01}a_{10}$$

dont la complexité est de 2 multiplications plus une addition.
Pour un déterminant d'ordre 3, on a :

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = a_{00}\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} - a_{10}\begin{vmatrix} a_{01} & a_{02} \\ a_{21} & a_{22} \end{vmatrix} + a_{20}\begin{vmatrix} a_{01} & a_{02} \\ a_{11} & a_{12} \end{vmatrix}$$

dont la complexité est de 3 multiplications, 2 additions, plus 3 fois la complexité du calcul d'un déterminant d'ordre inférieur. Par récurrence, on peut montrer que la complexité du calcul d'un déterminant d'ordre n est n multiplications, n - 1 additions plus n fois la complexité d'un déterminant d'ordre n-1. Par cumul, on arrive alors à *O(n . n!)*. Faisons l'hypothèse que l'ordinateur utilisé effectue une opération élémentaire en 10^{-9} seconde. On obtient alors les temps de calcul suivants pour les valeurs de n suivantes :

n	5	10	15
temps	$6 .10^{-7}$ s	0,04 s	5.45 heures
n	20	50	100
temps	1543 ans	$4,8 .10^{46}$ millénaires	$2,9594 . 10^{140}$ millénaires

Nous constatons que pour une certaine taille du problème, certains algorithmes ont un temps d'exécution inacceptable.

II.2.2.2 Classification des algorithmes

Habituellement la plupart des algorithmes admettent un paramètre de base n qui correspond à la taille du problème à résoudre. Cela peut être le degré d'un polynôme, le nombre d'enregistrements dans une base de données ou encore la profondeur d'un arbre. L'analyse de la complexité d'un algorithme reviendra à étudier l'incidence de l'augmentation de n sur la durée d'exécution du programme.

Les algorithmes habituellement rencontrés peuvent être classés dans les catégories suivantes :

- **Complexité constante O(1)**: On rencontre cette complexité quand toutes les instructions sont exécutées une seule fois pour n'importe quelle taille n du problème, par exemple le problème de parité d'un nombre.
- **Complexité logarithmique O(logn)**: La durée d'exécution croit légèrement avec n. Ce cas de figure se rencontre quand la taille du problème est divisée par une entité constante à chaque itération comme par exemple la recherche dichotomique.
- **Complexité linéaire O(n)**: C'est typiquement le cas d'un programme avec une boucle de 1 à n et le corps de la boucle effectue un travail de durée constante et indépendante de n, par exemple analyse des fréquences d'un texte à n caractères.

- **Complexité n-logarithmique O(nlogn)** : Se rencontre dans les algorithme où à chaque itération la taille du problème est divisée par une constante. Un exemple de ce genre de complexité est l'algorithme de tri "quick sort".
- **Complexité quadratique O(n^2)** : C'est le cas des algorithmes avec deux boucles imbriquées chacune allant de 1 à n et avec le corps de la boucle interne qui est constant.
- **Complexité cubique O(n^3)**: par exemple trois boucles imbriquées, avec les corps des deux boucles interne qui est constant.
- **Complexité exponentielle O(2^n)** : Les algorithmes de ce genre sont dits "naïfs" car ils sont inefficaces et inutilisable dès que n dépasse 50. On rencontre typiquement ces algorithmes dans les parcours arborescents.

II.2.2.3 Algorithme efficace

La complexité d'un algorithme est dite polynomiale si elle est $O(n^k)$, pour un certain entier k, on dit aussi, pour abréger, que l'algorithme lui-même est polynomial. Un algorithme est considéré comme efficace si et seulement si il est polynomial.

II.2.3 Complexité des problèmes

Dans cette section nous allons parler des problèmes faciles et difficiles et de la manière de les distinguer. L'expérience montre que certains problèmes sont plus difficiles que d'autres. Par exemple, étant donné un nombre, il est très facile de tester si ce nombre est pair ou non : il suffit de regarder si le dernier chiffre appartient à l'ensemble {0,2,4,6,8}. En revanche, il est beaucoup plus difficile de déterminer sa décomposition en produits de facteurs premiers.

La théorie de la complexité permet de classer les problèmes de décision (un problème de décision est un problème pour lequel la réponse est oui ou non) en deux classes faciles et difficiles en fonction de la complexité des algorithmes qui existent pour les résoudre.

Cependant il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil k correspondant à la fonction objectif f. Le problème de décision devient : "existe-t-il une solution réalisable (S) tel que $f(S) \leq k$ ($ou \geq k$)?".

Définition 2.3.1 *Un problème est dit facile s'il existe un algorithme polynomial pour le résoudre. Sinon le problème est dit difficile.*

Définition 2.3.2 *La complexité d'un problème est la complexité (dans le pire des cas) du meilleur algorithme connu pour le résoudre.*

II.2.4 Classes de complexité

- **La classe P (Polynomial)**: On dit qu'un problème est dans P s'il existe un algorithme polynomial en la taille des données pour le résoudre.

- **La classe NP (Non deterministic polynomial)**: Cette classe réunit les problèmes de décision pour lesquels la réponse oui peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille de l'instance (un algorithme est dit non déterministe s'il comporte des instructions de choix). Pour ces algorithmes, si à chaque instruction, le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps d'exécution devient exponentiel.

- **La classe Co-NP**: C'est le nom parfois donné pour l'équivalent de la classe NP, mais avec la réponse non. Pour montrer qu'un problème est dans NP, il suffit de trouver un algorithme qui vérifie si une solution donnée est valide en temps polynomiale.

- **La classe NP-Complet**: La classe NP-Complet est une sous-classe des problèmes NP. Un problème est NP-Complet quand tous les problèmes appartenant à NP lui sont polynomialement réductibles. Si on trouve un algorithme polynomial pour un problème NP-Complet, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe NP.

Définition 2.4.1 *Un problème A est réductible à un problème B s'il existe un algorithme résolvant A qui utilise un algorithme résolvant B.*

Définition 2.4.2 *Si l'algorithme résolvant A est polynomial, en considérant un appel à l'algorithme résolvant B comme une opération élémentaire, la réduction est dite polynomiale. On dit que A est polynomialement réductible à B.*

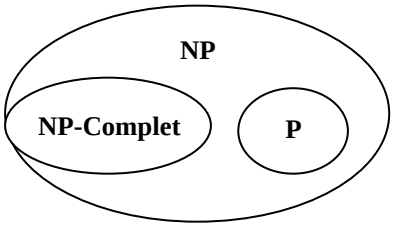


Figure 2.3 les différentes classes de complexité

II.2.5 Détermination de la classe d'un problème

Lorsque l'on est confronté à un problème P dont on ignore la classe de complexité, la démarche consiste soit :

A supposer que le problème est facile, et on cherche alors un algorithme polynomial pour le résoudre

A supposer que le problème est NP-Complet, et on cherche alors à le démontrer comme suit :

1. Choisir un problème P2, NP-Complet connu.
2. Construire une réduction f, qui transforme P2 vers P telle que:
   I une oui-instance de P2⇔ f(I) une oui-instance de P,
3. Prouver que f est une transformation polynomiale.

II.2.6 Conclusion

Nous venons de voir dans la partie I de ce chapitre, l'optimisation des requêtes par matérialisation des vues, nous avons également cité quelques travaux dans ce domaine et nous avons remarqué que la plupart des travaux s'intéressent au problème statique; ceci est dû essentiellement à la complexité du problème dynamique et au fait que le problème statique n'est pas complètement résolu.

Dans la deuxième partie nous avons vu la notion de problème et d'algorithme pour résoudre ce problème ainsi que la notion de complexité. Le problème que nous avons traité dans le cadre de cette thèse est NP-difficile, donc les méthodes exactes de la recherche opérationnelle sont inefficaces. Pour cela nous faisons appel aux metaheuristiques. Dans le chapitre suivant, nous allons présenter quelques metaheuristiques les plus connues.

Chapitre 3

Les Métaheuristiques pour l'optimisation combinatoire.

Objectif : *Dans ce chapitre nous allons parler des principales métaheuristiques qui existent sans vouloir être exhaustif. Nous parlerons de la descente, des algorithmes glouton, du recuit simulé , de la recherche tabou, des algorithmes génétiques, des colonies de fourmis et de l'optimisation par essaim de particules.*

3.1 Introduction

Il existe de nombreux problèmes d'optimisation combinatoire pour lesquels l'utilisation des méthodes exactes risquerait de prendre un trop grand temps de calcul. C'est le cas généralement des problèmes NP-difficiles. On se contente souvent d'une solution approchée, en essayant de faire en sorte qu'elle soit la meilleure possible dans un temps acceptable.

Il existe plusieurs méthodes qui permettent d'atteindre cet objectif. Certaines sont spécifiques à un problème, d'autres peuvent être adaptées à divers problèmes en précisant un certain nombre de paramètres. Ces méthodes sont appelées meta heurisdiques.

Une métaheuristique est constituée d’un ensemble de concepts fondamentaux qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation. Ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Les métaheuristiques sont représentées essentiellement par les méthodes de voisinage comme le recuit simulé et la recherche tabou, et les algorithmes évolutifs comme les algorithmes génétiques. Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications dont le traitement était impossible auparavant. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle **[HGH99].**

La grande vitalité de ce domaine de recherche ne doit pas faire oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable. Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème donné.

Toutes les métaheuristiques s'appuient sur un équilibre entre l'intensification et la diversification. L'intensification permet de rechercher des solutions de plus grande qualité en s'appuyant sur des solutions déjà trouvées. La diversification met en place des stratégies qui permettent d'échapper aux minimas locaux en explorant un plus grand espace de solutions.

Dans ce qui suit nous nous proposons de résoudre le problème suivant :

$$\begin{aligned} &Min\ f(X) \\ &X \in S \end{aligned}$$

Où S est un ensemble fini

Un élément de S s'appelle une solution réalisable. Un élément de S qui donne à f sa valeur minimale est appelé une solution optimale.

3.2 Les méthodes de descente

Le principe des méthodes de descente consiste, à partir d'une solution de départ X_0, à engendrer une suite finie de solutions X_i déterminées de proche en proche, X_{i+1} étant calculée à partir de X_i de telle sorte que X_{i+1} soit meilleure que X_i. On utilise la notion de voisinage pour engendrer la solution X_{i+1} à partir de X_i.

Le Voisinage d'une solution est obtenu en appliquant une transformation élémentaire (ou locale). Une transformation élémentaire ou locale est une opération qui permet de changer une solution X de S en une autre solution X' de S en ne modifiant que faiblement la structure de X. Par exemple si X est un entier codé en binaire, une transformation élémentaire consisterait à changer un bit de la chaîne en son complémentaire.

Etant donnée une transformation élémentaire, le voisinage V(X) d'une solution X est donc l'ensemble des solutions engendrées en appliquant à X cette transformation locale.

La notion de voisinage dépend de la transformation élémentaire appliquées. Pour l'exemple précèdent le voisinage d'une chaîne X de n bits est l'ensemble des n chaînes de n bits possédant exactement n-1 bits en commun avec X.

3.2.1 Schéma général d'une descente

Une descente consiste, à partir d'une solution initial, X_0 à engendrer une suite de solutions X_i vérifiant :

$$X_i \in V(X_{i-1}) \quad avec \quad f(X_i) < f(X_{i-1})$$

On arrête la méthode quand on atteint une configuration X_q telle que:

$$\forall X \in V(X_q), \quad f(X) \geq f(X_q)$$

X_q est minimum local de f par rapport à la transformation apportée

Schéma général :

Engendrer la Solution de départ X_0

$X_m=X_0$

Répéter

S'il existe $X \in V(X_m)$ tel que $f(X) < f(X_m)$ alors $X_m=X$

Jusqu'à ce qu'on ait $f(X) \geq f(X_m)$ pour tout $X \in V(X_m)$.

Retourner X_m comme solution finale

L'exploration du voisinage peut se faire de plusieurs manières :

- Lorsque la transformation dépend d'un paramètre, on tire aléatoirement ce paramètre et on applique la transformation. Il s'agit de l'exploration aléatoire. Dans cette exploration, il est difficile d'être sûr d'avoir atteint un minimum local pour cette transformation.
- L'autre exploration possible, est l'exploration systématique. Dans cette exploration, on envisage les uns après les autres tous les jeux de paramètres possibles, jusqu'à ce qu'on en trouve un qui conduise à un voisin meilleur que la solution courante. Si un tell voisin n'est pas trouvé c'est que la solution courante est un minimum local.
- La troisième exploration possible, est l'exploration exhaustive. Elle consiste à examiner les uns après les autres les voisins de la solution courante pour retenir celui qui entraîne la plus grande diminution de f.

La solution de départ peut être donnée par une autre méthode approchée ou générée de manière aléatoire. Dans ce dernier cas, on peut envisager d'appliquer plusieurs fois la méthode de la descente en changeant à chaque fois la solution de départ. Ceci peut atténuer l'inconvénient majeur de la descente qui réside dans le fait que la solution finale est un minimum local qui peut être loin du minimum global recherché.

3.3 Les Algorithmes glouton.

Le principe de ces algorithmes consiste à partir d'une solution incomplète, que l'on complète au fur et à mesure en effectuant des choix définitifs. C'est-à-dire qu'on ne remet plus en cause les choix pris en complétant la solution. Ces algorithmes sont souvent rapides mais la solution fournie peut être loin de la solution optimale. Ils peuvent être utilisés pour initialiser une méthode itérative.

3.4 Le recuit simulé

La méthode du recuit simulé repose sur une analogie avec la thermodynamique. Elle est issue d'études de l'évolution d'un système physique au contact d'un bain de chaleur. Elle à été proposée par S. Kirkpatrick et ses collègues [kir 83].

Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température est maintenu jusqu'à obtention d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires non satisfaisants, mais conduisant à la longue à des états meilleurs **[Mar 01].**

Le recuit simulé s'inspire de la méthode de la descente, mais au lieu de rejeter une transformation entraînant une augmentation de la fonction, on l'accepte avec une certaine probabilité, cette probabilité doit tenir compte de l'ampleur de l'augmentation et du degré d'avancement dans l'exécution. Un paramètre de contrôle (la température t) rendra de moins en moins probables les transformations désavantageuse. La probabilité d'accepter une transformation désavantageuse peut être choisie de la manière suivante :

On calcule la variation $\Delta f_{ij} = f(X_j) - f(X_i)$ de la fonction f et on accepte la transformation avec la probabilité p(i,j) suivante:

$$\begin{cases} p(i,j) = 1 & si \quad \Delta f_{ij} \le 0 \\ p(i,j) = e^{-\frac{\Delta f_{ij}}{t}} & si\ \Delta fij > 0 \end{cases}$$

Quand t est élevé (ce qui sera le cas initialement), on acceptera un grand nombre de transformations. Quand t atteindra une valeur basse proche de zéro, seule les transformations avantageuses seront prises en compte. Pour adopter une transformation désavantageuse, on tirera aléatoirement, avec une distribution uniforme un nombre p compris entre zéro et un. Si $p < e^{-\frac{\Delta f_{ij}}{t}}$ alors la transformation de X_i à X_j est acceptée. Sinon elle est refusée.

Le schéma général du recuit simulé est le suivant :

Calculer une solution initiale X

X_m=X

f_{min}=f(X_m)

Initialiser la température t

Tant que "*condition 1*" est vraie

Faire

(Dans cette grande boucle, on essaie toutes les transformations à une même température, puis on la fait décroître).

Tant que "condition 2" est vraie

Faire

(À chaque passage de cette boucle, on essaie une transformation et on détermine si elle est acceptée).

Choisir $X_1 \in$ voisinage (X)

Calculer $\Delta f = f(X_1) - f(X)$

Si Δf <0 (on adopte cette transformation faisant descendre)

Alors

X=X_1

Si f(X)<f(X_{min})

Alors

(On conserve la meilleure solution)

f(X_{min}) = f (X)

X_m=X

Fin si

Sinon

(On ne rejette pas systématiquement une solution qui augmenter f)

Tirer p dans [0,1] avec une distribution uniforme

Si $p \le e^{-\frac{\Delta f_{ij}}{t}}$ ***Alors*** X=X_1 Fin si

Fin si

Fait

t = g(t)

Fait Retourner X_m

La température décroît par palier jusqu'à ce que qu'elle soit suffisamment basse pour arrêter le processus. Ceci se traduit dans l'algorithme par la condition 1. Cette condition définit le nombre de changements de la température. Entre deux changements de température (donc à une température fixe) on effectue un certain nombre de transformations. Ce nombre est contrôlé dans l'algorithme par la condition2.
Dans cette description, il y'a lieu de fixer six paramètres.

- Le choix de la solution initiale
- Le choix de la condition1
- Le choix de la condition 2
- La valeur initiale de la température
- La décroissance de la température
- La transformation.

Ces paramètres doivent être fixés par l'utilisateur selon sa connaissance du problème, du temps de calcul acceptable, de la qualité de la solution souhaitée. Néanmoins, nous pouvons donner quelques indications sur ces choix.

1. ***Solution initiale****:* La solution initiale peut être prise de manière aléatoire, ou bien fournie par un algorithme glouton. Une bonne solution de départ permet de démarrer à une température plus basse, faisant gagner du temps d'exécution. Si on ne connaît rien sur la qualité de la solution, on devra démarrer à des températures assez élevées.
2. ***Valeur initiale de la température t*** *:* La température initiale doit être suffisamment grande pour que de nombreuses transformations soient acceptées. On peut démarrer à une certaine température élevée et essayer un certain nombre de transformations, puis calculer le taux d'acceptation τ qui est le rapport entre le nombre de solutions désavantageuses acceptées sur le nombre de solutions engendrées. Si ce taux est assez élevé, (>50 % par exemple) on adopte cette température, sinon on la double et on recommence jusqu'à ce que la valeur τ convienne. Si on exige un taux d'acceptation initiale trop élevé, on risque d'augmenter considérablement le temps de calcule.

3. ***Décroissance de la température*** *:* On peut utiliser comme modèle pour la décroissance de t, la fonction g définie par g(t)=μ.t avec 0<μ<1. La suite des températures est donc une suite géométrique tendant vers zéro. Le paramètre μ peut être fixé par expérimentation. En général, il est pris proche de 1 (0,85 par exemple). Nous pouvons aussi lier μ au degré d'avancement dans l'algorithme pour accroître ou réduire la vitesse de décroissance.
4. ***Condition 1:*** Cette condition détermine le nombre de changements de températures effectués au cours de l'algorithme. Il faut qu'à la fin de l'algorithme la température soit suffisamment faible pour qu'aucune solution désavantageuse ne soit acceptée. On peut aussi s'arrêter lorsque le gain sur f est faible pour continuer. Ou si on n'améliore plus la solution pendant un certain nombre d'itérations.
5. ***Condition2****:* Cette condition détermine le nombre d'essaies de transformations à température fixe. On peut retenir comme condition : On ne modifie pas t tant qu'on n'a pas accepté un certain nombre de transformations ou qu'on n'a pas atteint un certain nombre d'itérations à température fixe.
6. ***Transformation****:* la transformation adoptée doit engendrer tout l'espace des configurations sans requérir trop d'opérations pour sa mise en œuvre.

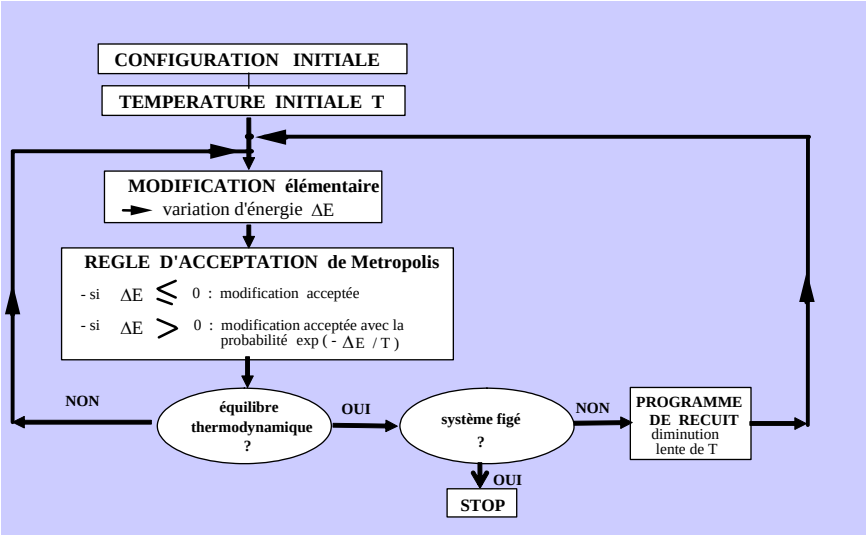


Figure 3.1 organigramme général du recuit simulé

3.5 La recherche tabou

La recherche tabou a été présentée par Fred Glover dans son article paru en 1989 **[Glo 89]**. Elle est essentiellement axée sur une exploration de l'ensemble des solutions en utilisant la notion de voisinage.

Formellement, on définit pour toute solution s de S un ensemble N(s) $\subset$ S appelé ensemble des voisins de s

Les méthodes de recherche locales, prennent une solution à un problème et essayent de modifier légèrement cette solution jusqu'à ce qu'il ne soit plus possible d'améliorer cette solution. En d'autres termes, l'arrêt se fait à la rencontre d'un optimum local. Et rien n'indique que c'est un optimum global.

Afin de pouvoir trouver des solutions meilleures, que le premier optimum local rencontré, on peut essayer de poursuivre le processus de modifications locales. Cependant si on ne prend pas des précautions, on risque de reconsidérer cycliquement un nombre restreint de solutions. La recherche tabou tente de remédier à ce problème en interdisant le retour à l'optimum local déjà trouvé et ce en rendant des solutions taboues.

A la base des recherches locales il y'a la définition de l'ensemble N(s) des solutions voisines de s, mais d'un point de vue pratique, on considère l'ensemble des mouvements que l'on peut apporter à s plutôt que N(s).

L'ensemble N(s) s'exprime comme l'ensemble des solutions que l'on peut obtenir de s en appliquant un mouvement m appartenant à l'ensemble M des mouvements. L'application de m à s sera notée s $\oplus$ m et donc N(s)={s'/s'= s $\oplus$ m, m$\in$M}.

Dans une recherche locale, on ne considère pas forcement à chaque itération l'ensemble des solutions de N(s) mais seulement un sous ensemble.

L'idée de départ de la méthode consiste à se déplacer de solution en solution en s'interdisant de revenir à une configuration déjà rencontrée. Soit V(X) le voisinage de X et soit T la liste de toutes les configurations rencontrées depuis le début. De l'exécution. A partir de la configuration courante de X, on choisi dans V(X)-T, la configuration Y qui minimise la fonction objectif f, tout en ajoutant X à le liste T.

En pratique, au lieu de conserver la solution X dans T, on conserve le mouvement qui a permis de passer de la configuration courante à la suivante et on interdit d'appliquer son inverse. Ce sont donc les mouvements qui sont interdits et non plus des solutions.

Pour ne pas trop appauvrir le voisinage de la configuration courante, on limite la taille de la liste T que l'on gère comme une file d'attente FIFO.
Tant que la file n'est pas pleine, on ajoute les mouvements interdits. Quand la file est pleine, on supprime de T le mouvement qui s'y trouve depuis le plus longtemps, et on le remplace par le nouvelle modification à interdire. Le choix de la taille de T est très important, car une grande taille appauvrit le voisinage et une petite taille risque de ne pas pouvoir empêcher le bouclage. Un des schémas de la méthode tabou est le suivant :

Calculer une solution initiale X
$X_m=X$; $f_{min}=f(X_m)$;$T=\varnothing$;Nb-iter=0

Tant que Nb-iter <nbmax
Faire
Nb-iter=Nbiter+1
C=V(X)-{m(X) pour m∈T} (ensemble des configurations candidates).
Déterminer Y=m(X)qui minimise f sur C(Y est l'un des meilleurs voisins de X)
Si f(Y) ≥f(X) ***alors*** mettre à jour T (on ajoute m^{-1} à T au besoin en éliminant l'élément le plus ancien de T)
Si f(Y)<fmin
Alors $X_{m=Y}$
$F_{min}=f(Y)$
Fin si
X=Y
Fait

La recherche tabou présente donc les caractéristiques suivantes:

- Elle est déterministe ,
- Elle possède une mémoire,
- Elle a un nombre de paramètres à régler réduit et
- Elle explore un échantillon du voisinage.

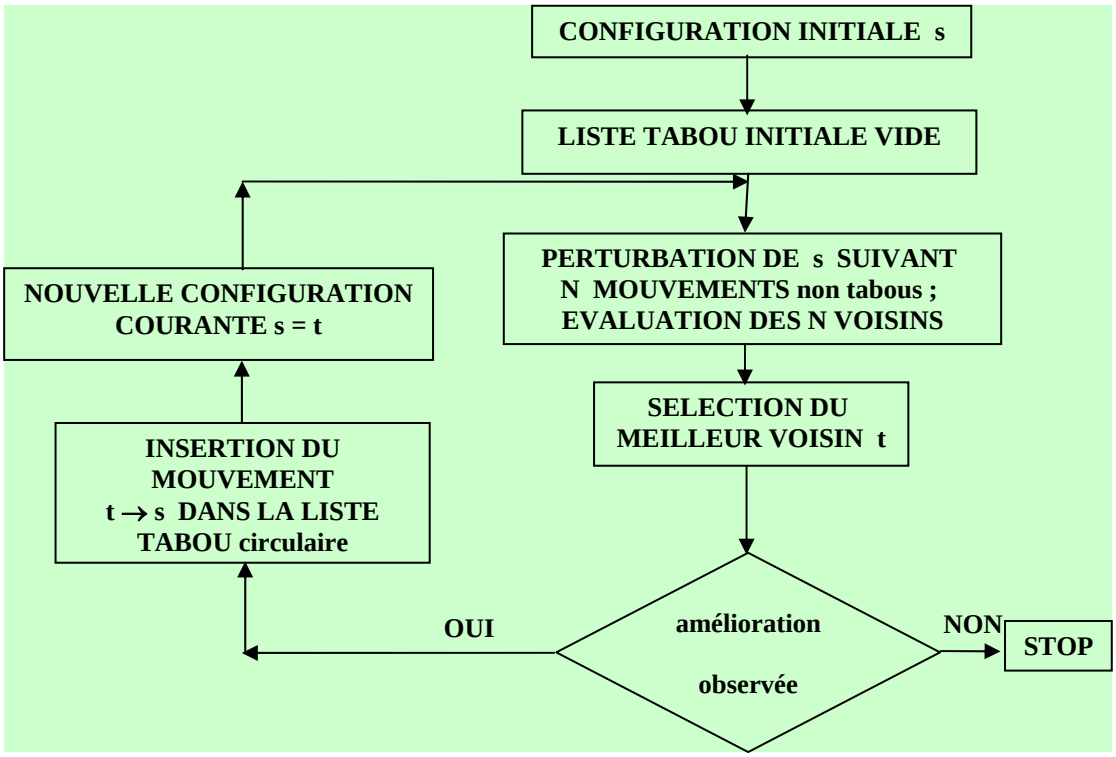


Figure 3.2 organigramme général de la recherche tabou

3.6 Les algorithmes génétiques

Les algorithmes génétiques ont été développés par J. Holland [Hol 92] et son équipe, à l'université du Michigan. Leurs recherches avaient pour buts principaux

- Mettre en évidence et expliquer rigoureusement les processus d'adaptation des systèmes naturels et
- Concevoir des systèmes artificiels qui possèdent les propriétés importantes des systèmes naturels

Ces algorithmes sont des algorithmes d'exploration fondés sur des mécanismes de la sélection naturelle et de la génétique. Ils utilisent à la fois les principes de la survie des structures les mieux adaptées, et les échanges d'information pseudo aléatoires pour former un algorithme d'exploration qui possède certaines des caractéristiques de l'exploration humaine.

A chaque génération, un nouvel ensemble de créatures artificielles est créé en utilisant des parties des meilleurs éléments de la génération précédente.

Contrairement à la plupart des autres méthodes, les algorithmes génétiques ne considèrent pas une seule configuration à la fois mais une population de configurations $X_1, X_2, \ldots, X_p$ appelée population où p est la taille de la population. (p peut être fixe ou variable).

Une solution X_i est représentée et manipulée sous la forme d'une chaîne de caractères Qu'on appelle, par analogie à la génétique, chromosome et gènes les caractères la constituant.

Trois opérateurs caractérisent le algorithmes génétiques:

- La sélection,
- Le croisement et
- La mutation.

Ces trois opérateurs permettent à la population d'évoluer par la construction d'individus nouveaux à l'aide d'anciens. On puise dans certains individus de la population courante, une partie de leurs caractéristiques en prélèvant certaines parties des chromosomes qui les représentent, puis on recombine ces différentes parties pour constituer les individus de la nouvelle population.

- La sélection indique dans quelles configurations de la population courante on va prélever les morceaux de chromosomes.
- Le croisement combine les morceaux de chromosomes pour former les configurations de la population suivante.
- La mutation change éventuellement certains gènes de certains chromosomes des nouveaux individus.

La succession de ces trois opérations constitue une génération.

3.6.1 La sélection

Cet opérateur sélectionne les individus sur lesquels on va appliquer le croisement. Il doit dépendre de la fonction objectif f à minimiser. L'individu X_i aura plus de chance d'être choisi si $f(X_i)$ est plus petite.

Une des façons de faire intervenir f dans le sélection, est de l'utiliser dans la probabilité de sélectionner X_i qui peut être donnée par

$$p = \frac{F - f(X_i)}{F(p-1)}$$

$$où\ F = \sum_{i=1}^{p} f(X_i)$$

p *est le nombre d'individus*

Donc meilleur est l'individu, plus grande est la probabilité. pour qu'il soit sélectionné. Une fois les probabilités de sélection fixées, on applique ces probabilités à un ensemble d'individus de la population. Ces individus seront croisés à une probabilité p_c.

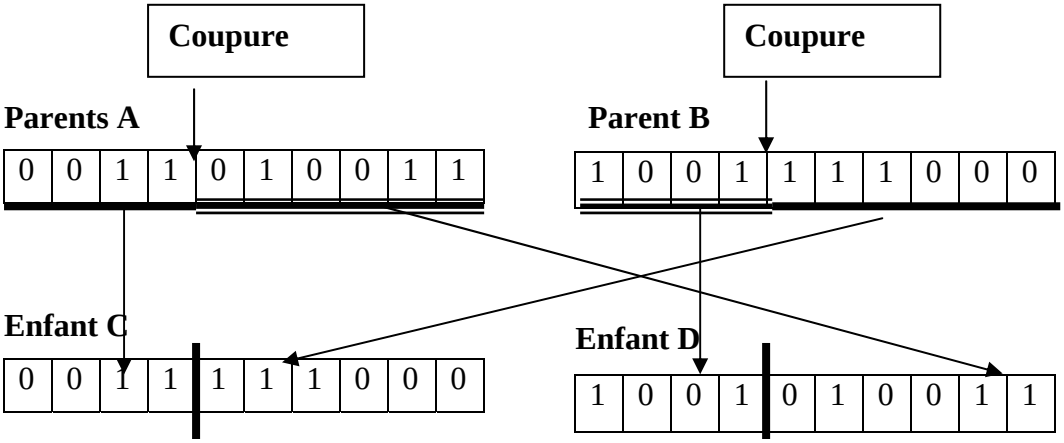
3.6.2 Le croisement

le but du croisement est de recombiner d'une certaine façon les chromosomes de deux parents afin de former les chromosomes des deux enfants. Un croisement que l'on rencontre souvent est le croisement à un point. Ce type de croisements fait que les couples de gènes n'ont pas tous la même probabilité de rester ensemble (les extrémités du chromosome seront toujours séparés).

On peut modifier ce croisement en un croisement en deux points (ou k points), au lieu de tirer aléatoirement une position après laquelle on fait la coupure, on tire aléatoirement deux positions (k positions), et on échange les morceaux de chromosomes compris entre ces deux points de coupure.

Exemple:

Croisement en un point



Croisement en deux points (ou en k points)

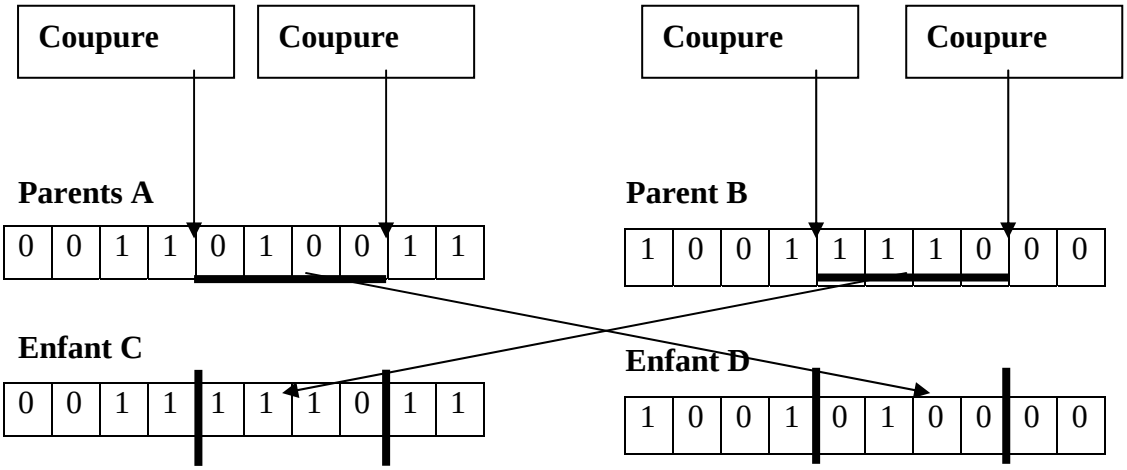


Figure 3.3 croisement en un point et en deux points

Ce procédé présente l'avantage de conserver les caractéristiques communes. Si les deux parents possèdent un gène identique au même endroit, les enfants le possèderont aussi, ce qui est souhaitable.

Cette propriété est aussi vérifiée par le croisement uniforme qui consiste pour chaque gène du chromosome que l'on construit, à choisir aléatoirement le parent qui va donner son gène pour pourvoir cette position.

3.6.3 La mutation

Cet opérateur est conçu pour apporter une certaine diversité dans la population et empêcher que celle-ci ne converge trop vite vers un même individu ou vers un petit groupe d'individus.

Il agit aléatoirement sur le codage d'un individu, en remplaçant un ou plusieurs des symboles du codage par autant d'autres symboles de l'alphabet. Par exemple, dans le cas d'un entier codé en binaire, la mutation peut être de changer un 1 par 0 ou un 0 par un 1. Pour ne pas trop perturber la composition de la population et ne pas transformer les algorithmes génétiques en une errance aléatoire, la mutation doit avoir une faible probabilité. p_m. On peut avoir une mutation par chromosome ou une mutation par gène.

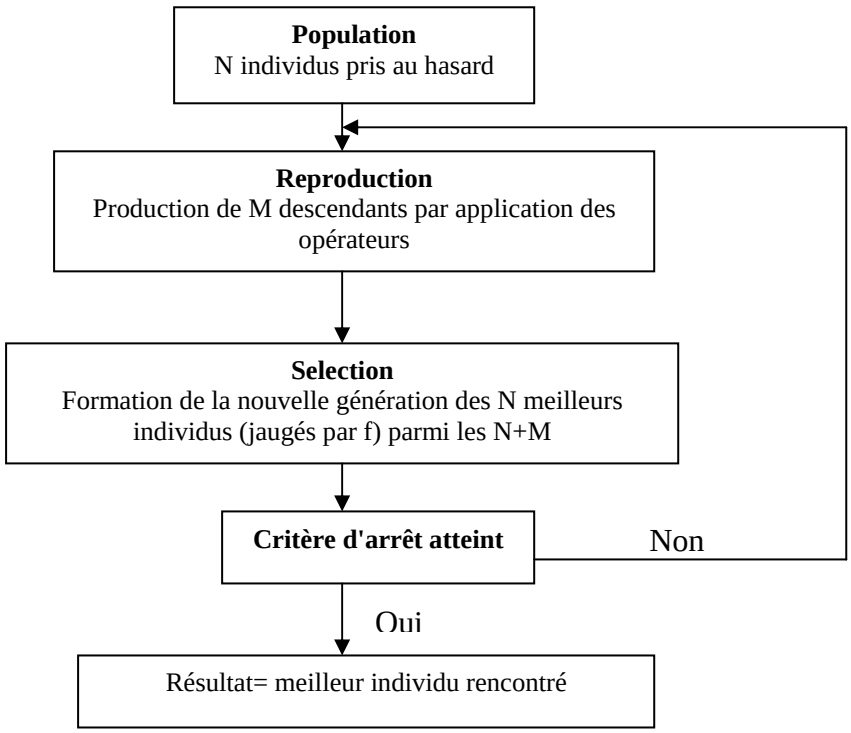


Figure 3.4 Organigramme général des algorithmes génétiques

3.7Algorithme des Colonies de fourmis

3.7.1 introduction

Lorsqu'un nouveau problème se pose, il faut parfois définir de nouvelles méthodes de résolution , car les anciennes peuvent ne plus être adaptées au cas traité. Souvent, il faut une source d'inspiration qui peut être issue de la modélisation des systèmes naturels.

Il s'agit de copier et d'adapter les concepts naturels pour la résolution de problèmes. La biologie constitue une source d'inspiration qui a de plus en plus de succès en informatique dans une branche que l'on peut nommer l'informatique biomimétique. Le principe de base de cette méthode de développement suit le schéma de Arkin **[Ark 98]**

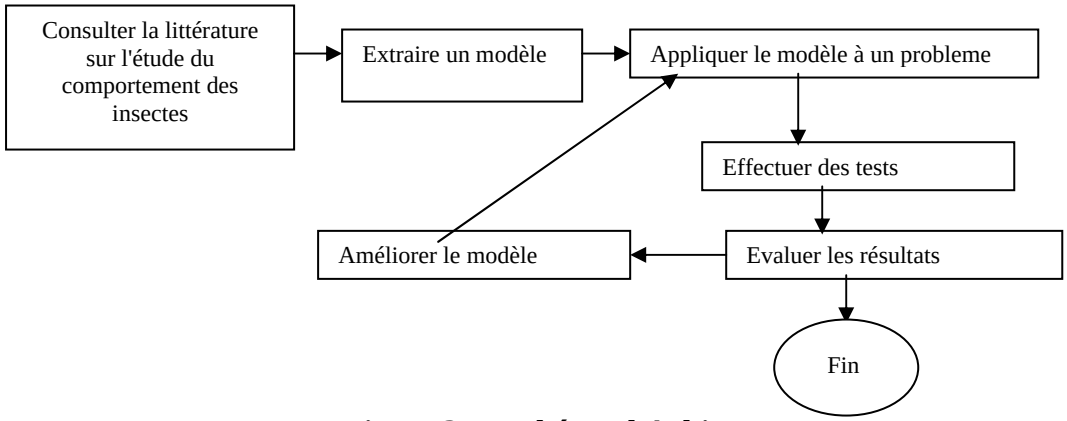


Figure 3.5 : schéma d'Arkin

Le comportement des insectes sociaux est caractérisé par l'auto organisation. Les individus communiquent en changeant les propriétés locales de leur environnement et par le biais de ce moyen de communication limité, une sorte d'intelligence collective émerge **[Seg03].**

La biologie est la source d'inspiration de nombreuses métaheuristiques. Ainsi, les théories de l'évolution ont inspiré les algorithmes évolutionnaires **[Gol 94]**, les phénomènes de suivi de piste chez les fourmis ont conduit à l'élaboration des algorithmes de colonies de fourmis **[BDT 99]**, l'étude de l'organisation de groupes d'animaux a donné naissance aux méthodes d'optimisation par essaim particulaire **[EKS 01]**. Il existe, en outre, d'autres algorithmes, moins connus que ceux que nous venons de citer, qui découlent de la biologie : en particulier, des algorithmes inspirés du fonctionnement du système immunitaire **[DeV 99]**

des algorithmes pour l'allocation dynamique de tâches **[CiS 01]**, s'appuyant sur des modèles d'organisation du travail chez les fourmis, des algorithmes de classification suggérés par les essaims d'insectes **[AMS 03]**. Une contribution importante de la biologie dans ce domaine vient de la théorie de l'auto organisation **[CDF 00]**, qui permet d'analyser les propriétés de plusieurs métaheuristiques issues des métaphores biologiques.

L'intelligence en essaim est ainsi née sur deux fronts : via une approche "systèmes auto organisés" (ayant donné lieu aux algorithmes de colonies de fourmis) et via une approche "systèmes sociocognitifs" (ayant conduit à l'optimisation par essaim particulaire).

3.7.2 Auto organisation

L'auto organisation est un phénomène décrit dans plusieurs disciplines, notamment en physique **[PrG 71]** et en biologie

. *L'auto organisation caractérise un processus au cours duquel une structure émerge au niveau global uniquement d'un grand nombre d'interactions entre les composants de niveau local du système. De plus, les règles spécifiant les interactions entre composants du système sont suivies en utilisant uniquement des informations locales, sans référence au modèle global.* **[CDF 00]**

Deux termes sont à préciser pour une bonne compréhension, "structure" et "émerger". Le mot structure est une traduction approximative du mot anglais "pattern", qui déborde la notion de modèle, et peut signifier aussi configuration générale, forme, schéma, type. D'une manière générale, il s'applique à un "arrangement organisé

d'objets dans l'espace ou le temps". Une propriété émergente d'un système est quant à elle une caractéristique qui apparaît "à l'improviste" (sans avoir été explicitement déterminée), de par les interactions entre les composants de ce système. Nous parlerons donc d'émergence pour souligner le caractère non déterminé d'une propriété, sans faire référence au fait que cette propriété soit organisée en structures, ni qu'elle soit située à un niveau différent des interactions.

La question cruciale est donc de comprendre comment les composants d'un système interagissent entre eux pour produire une structure complexe (au sens relatif du terme, i.e. plus complexe que les composants eux-mêmes). Un certain nombre de phénomènes nécessaires ont été identifiées: ce sont les processus de rétroaction et la gestion des flux d'informations.

Les rétroactions positives sont des processus dont le résultat renforce l'action, par exemple par amplification. Les rétroactions positives sont capables d'amplifier les fluctuations du système, permettant la mise à jour d'informations peu apparentes. De tels processus peuvent facilement entraîner une "explosion" du système, s'ils ne sont pas maintenus sous contrôle par des rétroactions négatives, qui jouent ainsi le rôle de stabilisateurs du système. **[CDF 00]**

Dans le cadre de la biologie du comportement, il est aisé de comprendre que les interactions entre les composants d'un système vont très souvent mettre en jeu des processus de communication, de transfert d'informations entre individus. D'une manière générale, les individus peuvent communiquer, soit par le biais de signaux, c'est à dire en utilisant un moyen spécifique pour porter une information, soit par le biais d'indices, où l'information est portée accidentellement. De même, l'information peut provenir directement d'autres individus, ou bien passer par le biais de l'état d'un travail en cours. Cette deuxième possibilité d'échange des informations, par le biais de modifications de l'environnement, se nomme la stigmergie

D'une manière générale, tous ces processus sont plus ou moins interconnectés, permettant à un système constitué d'un grand nombre d'individus agissant de résoudre des problèmes trop complexes pour un individu unique. Certaines caractéristiques des systèmes auto organisés sont particulièrement intéressantes, en particulier leur dynamisme, ou encore leur capacité à produire des modèles stables. Dans le cadre de l'étude du comportement des insectes sociaux, certains concepts liés au principe de l'auto organisation méritent d'être soulignées : la décentralisation intrinsèque de ces

systèmes, leur organisation en hétérarchie dense et l'utilisation récurrente de la stigmergie. En effet, ces concepts sont parfois utilisés comme autant d'angles de vue différents sur un même problème et recouvrent une partie des principes de l'auto organisation.

3.7.3 Stigmergie

La stigmergie est un des concepts à la base de la création des métaheuristiques de colonies de fourmis. Elle est précisément définie comme une "forme de communication passant par le biais de modifications de l'environnement", mais on peut rencontrer le terme "interactions sociales indirectes" pour décrire le même phénomène. La spécificité de la stigmergie est que les individus échangent des informations par le biais du travail en cours, de l'état d'avancement de la tâche globale à accomplir.

3.7.4 Contrôle décentralisé

Dans un système auto organisé, il n'y a pas de prise de décision à un niveau donné, suivie d'ordres et d'actions prédéterminées. En effet, dans un système décentralisé, chaque individu dispose d'une vision locale de son environnement, et ne connaît donc pas le problème dans son ensemble. La littérature des systèmes multi agents **[Fer 97]** pour une première approche) emploie souvent ce terme ou celui "d'intelligence artificielle distribuée" **[Jen 96]**, bien que, d'une manière générale, l'étude des systèmes multi agents tende à utiliser des modèles de comportement plus complexes, fondés notamment sur les sciences de la cognition. Les avantages d'un contrôle décentralisé sont notamment la robustesse et la flexibilité: systèmes robustes, car capables de continuer à fonctionner en cas de panne d'une de leurs composantes; flexibles, car efficaces sur des problèmes dynamiques

3.7.5 Hétérarchie dense

L'hétérarchie dense est un concept issu directement de la biologie, utilisé pour décrire l'organisation des insectes sociaux, et plus particulièrement des colonies de fourmis. Elle peut être définie comme suit :

Une colonie de fourmis est une variante particulière de hiérarchie qui peut avantageusement être appelée une hétérarchie. Cela signifie que les propriétés des

niveaux globaux agissent sur les niveaux locaux, mais que l'activité induite dans les unités locales influence en retour les niveaux globaux.

L'hétérarchie est dite dense dans le sens où un tel système forme un réseau hautement connecté, où chaque individu peut échanger des informations avec n'importe quel autre. Ce concept est en quelque sorte opposé à celui de hiérarchie où, dans une vision populaire mais erronée, la reine gouvernerait ses sujets en faisant passer des ordres dans une structure verticale, alors que, dans une hétérarchie, la structure est plutôt horizontale (figure 3.6).

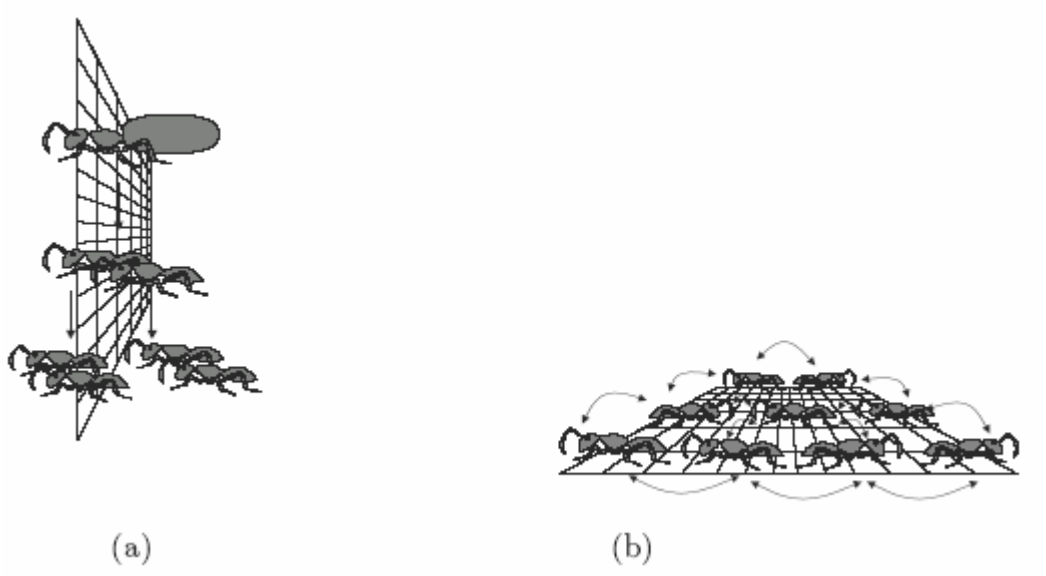


Figure 3.6 Hiérarchie (a) et hétérarchie dense (b) : deux concepts opposés.

On constate que ce concept recoupe celui de contrôle décentralisée, mais aussi celui de stigmergie, en ce sens que l'hétérarchie décrit la manière dont le flux d'information parcourt le système. Cependant, dans une hétérarchie dense, tout type de communication doit être pris en compte, tant la stigmergie que les échanges directs entre individus.

Les fourmis naturelles ont inspiré les « algorithmes à colonies de fourmis », l'optimisation par colonies de fourmis (ACO) est une technique d'optimisation biomimétique (c'est-à-dire dans les sciences qui s'inspirent du vivant) inspiré par un travail de biologiste en 1983, repris par des informaticiens en 1988 et largement exploité et développé par Marco Dorigo dans les années 90. L'idée consiste à imiter le comportement des fourmis réelles qui collaborent, par exemple pour la recherche de

sources de nourriture en mélangeant comportement d'exploration aléatoire et suivi des traces chimiques laissées par leur consoeurs. Ces traces chimiques, les « phéromones », sont utilisées par les fourmis pour communiquer entre elles de manière indirecte, par le biais de l'environnement, une technique générale connue par les entomologistes sous le nom de stigmergie. C'est cette forme de communication ainsi que l'idée de faire coopérer une foule d'agents simples et localisés qui forme la base de l'heuristique développée par Dorigo.

D'abord appliquée aux problème du voyageur de commerce, l'optimisation par colonies de fourmis a rapidement prouvé son efficacité dans le cadre de l'optimisation combinatoire en général et s'est montré particulièrement profitable pour le problème du routage des paquets d'information dans les grands réseaux d'interconnexion. L'optimisation par colonies de fourmis forme aujourd'hui un champ de recherche à part entière.

3.7.6 Principe de fonctionnement des colonies de fourmis

Pour mieux comprendre la façon dont fonctionnent les colonies de fourmis, prenons l'exemple traditionnellement donné pour illustrer leur capacité à trouver des chemins optimaux. La figure 3.7 illustre une situation où il y a un nid, où les fourmis vivent, et une source de nourriture, que les fourmis doivent trouver et dont elles doivent ramener les provisions vers le nid **[Yan03]**.

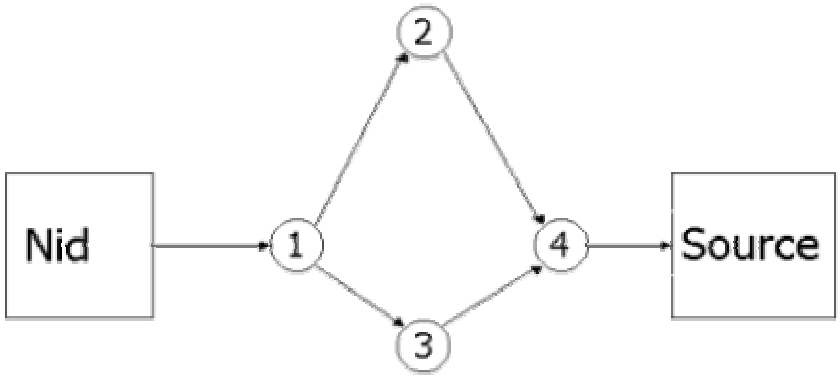


Figure 3.7 expérience du nid des fourmis et de la source de nourriture

Lorsqu'une ouvrière chargée de rechercher de la nourriture quitte le nid, elle va déposer sur son chemin, à l'aide d'un aiguillon placé sur son abdomen, une fine trace

de phéromone d'orientation. Lorsqu'elle retournera au nid après avoir trouvé une source d'approvisionnement, elle déposera à nouveau des phéromones, un peu différentes puisqu'ici une information sur la qualité du site trouvé sera incluse dans ce message. Dans tous les cas, la phéromone est placée là pour les autres fourmis de la colonie afin de les inciter à choisir cette piste.

Ce marquage leur permet également de trouver le plus court chemin entre le nid et un site alimentaire. En effet, au départ chaque fourmi choisira l'une des pistes disponibles totalement au hasard. Très rapidement le chemin le plus court sera marqué par plus de phéromone, car le va-et-vient des fourmis y est plus rapide, et pour une même période, plus de fourmis ont le temps de le parcourir, donc de déposer leur message. Les nouvelles fourmis qui quittent le nid ont une tendance naturelle à favoriser la piste qui comporte la trace olfactive la plus importante **[ROU 01].**
Cette façon de procéder est déjà efficace en soi mais il manque deux phénomènes essentiels pour qu'elle soit tout à fait complète.

Le premier phénomène est qu'il arrive quelquefois que des fourmis étourdies se trompent et s'écartent du chemin de phéromones. Si, par chance, une fourmi égarée par erreur trouve un chemin plus court, la trace de phéromone qu'elle laissera derrière elle sera plus fraîche, indiquant par là même aux autres fourmis qu'il existe un chemin plus court pour accéder à la nourriture. Ainsi, c'est le mécanisme d'erreur dans le suivi de trace de phéromone qui permet la découverte de raccourcis, aboutissant à l'établissement d'un chemin optimal entre fourmilière et nourriture.

Le deuxième phénomène est que les phéromones s'évaporent dans le temps, rendant ainsi leurs traces éphémères. C'est ce deuxième mécanisme qui permet aux chemins établis de ne pas être statiques, et de s'adapter aux modifications de l'environnement. Supposons maintenant que seul le chemin le plus long soit disponible (une brindille obstrue le chemin court). Les fourmis vont donc l'utiliser et le charger en phéromones. Supposons maintenant que le chemin redevienne tout à coup disponible (la brindille a été poussée par le vent), notre colonie se trouve dans une situation non optimale : une piste de phéromones encourage les fourmis à suivre un chemin qui n'est pas le meilleur puisqu'elles suivent le chemin long alors que le court est à nouveau

disponible. La situation est rétablie par le caractère volatil des phéromones : sans apport de phéromone important, le chemin le plus long finit par s'effacer pour disparaître presque complètement. Grâce à l'action conjuguée de l'évaporation et de la croissance rapide de la concentration en phéromones sur le chemin court, le chemin long va rapidement tomber en désuétude et l'optimalité sera rétablie.

3.7.7 Les fourmis artificielles

La fourmi artificielle se présente sous la forme d'un ensemble de procédures qui définissent son comportement. Celui-ci est très semblable à celui de la fourmi naturelle quand elle recherche de la nourriture. Dans ce cas, une fourmi n'a qu'un rôle assez simple qui consiste à se déplacer du nid jusqu' à la source de nourriture et à y revenir.

Le code qui définit le comportement des fourmis artificielles permet de se déplacer dans l'espace combinatoire formé par les différents éléments qui peuvent être utilisés pour le problème à résoudre, c'est à dire pour construire une solution.

3.7.8 La métaheuristique ACO (optimisation par colonies de fourmis)

Les fourmis artificielles utilisées dans la métaheuristique ACO (Ant Colony Optimization) sont des procédures stochastiques qui construisent des solutions en ajoutant itérativement une nouvelle solution à l'ensemble des solutions existant en tenant compte de l'information heuristique et de la phéromone artificielle.

L'ACO peut être interprétée comme une extension d'une heuristique traditionnelle de recherche, qui peut être appliquée à plusieurs problèmes d'optimisation combinatoire. Ce qui fait la différence entre l'ACO et les heuristiques traditionnelles c'est l'adaptation de la phéromone durant l'exécution de l'algorithme pour tenir en compte de l'expérience de la recherche. L'interprétation de l'ACO comme une heuristique de construction extensible est évidente pour plusieurs raisons, le composant stochastique dans l'ACO permet aux fourmis de construire une large variété de solutions, au même moment l'utilisation de l'information heuristique peut guider les fourmis vers les meilleures solutions. De plus l'expérience de la recherche des fourmis est utilisée

pour influencer dans un chemin évocateur la construction de la solution dans les futures itérations de l'algorithme **[Dor01]**.

3.7.8.1 Représentation d'un problème par les algorithmes ACO

Soit le problème de minimisation suivant *(S, f, Ω)*, où *S* représente l'ensemble des solutions candidates, *f* la fonction objectif associée à chaque solution $s \in S$. *f (s, t)* est la fonction objectif avec le paramètre temps t dans le cas ou la fonction objectif dépend du temps (par exemple le cas d'une application dynamique), et Ω est la contrainte. L'objectif est de trouver la solution optimale $S_{opt} \in S$ avec un coût minime et qui respecte la contrainte Ω.

Le problème de représentation d'un problème d'optimisation combinatoire (*S, f,* Ω) exploité par les fourmis peut être caractérisé comme ceci :

On donne l'ensemble fini $C = \{c_1, c_2, \ldots, c_{nc}\}$ de composants, *nc* étant le nombre de composants. (l'ensemble C représente les composants du problème)

L'état du problème est définie en terme de séquences $x = \{c_i, c_j, \ldots, c_k, \ldots\}$ sur les éléments de C. l'ensemble de toutes les séquences possibles est noté par χ. La taille de la séquence *x* est le nombre de composants dans la séquence, noté $| x |$.

L'ensemble finit des contraintes Ω définit l'ensemble des états faisables χ', avec $\chi' \subseteq \chi$.

L'ensemble S* des solutions faisables est donné par $S^* \subseteq \chi'$ et $S^* \subseteq S$.

Le coût *f(s, t)* est associé à chaque solution candidate $s \in S$.

Etant donné ces représentations, les fourmis artificielles construisent les solutions en parcourant le graphe $G = (C, L)$, où les nœuds sont les composants de *C*, et *L* est l'ensemble des arcs qui connectent complètement les composants de *C*. Le problème des contraintes Ω est implémenté dans une politique suivie par les fourmis artificielles.

3.7.8.2 Le comportement des fourmis dans l'ACO

Les fourmis peuvent être caractérisées comme des procédures de constructions stochastiques, qui construisent des solutions en parcourant le graphe $G = (C, L)$. Les fourmis se déplacent dans le graphe en suivant une politique de construction qui est une fonction du problème et de la contrainte Ω.

En général, les fourmis essaient de construire les solutions faisables, mais si c'est nécessaire elles peuvent générées des solutions infaisables. Une phéromone peut être associée aux composants $c_i \in C$ et aux connecteurs $l_{ij} \in L$. τi est la phéromone associée aux nœuds et τ_{ij} est la phéromone associée à l'arc allant du nœud i vers le nœud j). Une valeur heuristique η peut aussi être associée aux nœuds et aux connecteurs (η_i pour les nœuds, et η_{ij} pour les arcs) cette valeur représente en général le coût. Ces valeurs (phéromone et la valeur heuristique) sont utilisées par la fourmi pour définir la probabilité du déplacement dans le graphe.
Plus précisément chaque fourmi k de la colonie a les propriétés suivantes :
Elle exploite le graphe $G = (C, L)$ pour chercher les solutions faisables avec le coût minimal.
Elle a une mémoire M_k qu'elle utilise pour enregistrer les informations sur le chemin parcouru précédemment. La mémoire peut être utilisée pour :

- Construire les solutions faisables.
- Evaluer la solution trouvée.
- Retracer le chemin pour déposer la phéromone.

On peut attribuer un état initial x^k_s (pour une fourmi k et pour une solution s) et une ou plusieurs conditions de terminaisons e_k. Généralement, l'état initial d'une séquence contient un seul composant ou une séquence vide.
Lorsqu'elle est dans l'état $x_r = < x_{r-1}, i >$ (ou x_{r-1} est l'état précèdent et i est le nœud ou elle se trouve), si aucune condition d'arrêt n'est pas satisfaite, elle se déplace vers le nœud j (qui appartient à ses voisins non encore visités), son état devient $< x_r, j > \in \chi$.

Souvent, les déplacements vers les solutions faisables sont favorisés, soit par la valeur de l'heuristique η, ou à travers l'utilisation de la mémoire des fourmis.
Le déplacement entre les nœuds se fait par une probabilité de déplacement. Cette dernière inclue :

- Le taux de phéromone τ.
- La valeur heuristique η.
- Sa mémoire M_k.
- La contrainte du problème.

La procédure de construction d'une fourmi k s'arrête lorsque une des conditions d'arrêt e_k est satisfaite.
Lorsqu'un composant (nœuds) est ajouté à l'ensemble des solutions, la fourmis met à jour la phéromone qui lui est associée ou au connecteur correspondant. Ceci est appelé mise à jour local de la phéromone ou mise à jour pas à pas en ligne (on line step-by-step phéromone update).
Une fois la solution est construite, elle peut retracer le même chemin et mettre à jour la phéromone associée aux composants ou aux connecteurs sélectionnés. Ceci est appelé mise à jour de la phéromone différée en ligne (on line delayed pheromone update).
Il est important de préciser que les fourmis se déplacent de manière concurrente et indépendante et chaque fourmi trouve une solution. Une bonne solution émerge en résultat de l'interaction collective des fourmis **[Dor01]**.

3.7.8.3 Exemple d'adaptation des colonies de fourmis à un problème

Le premier exemple des algorithmes à base de colonies de fourmis (ACO) est le AS (Ant System « système de fourmis ») conçu pour le problème de voyageur de commerce **[Dor01]**. Pour illustrer la manière dont on peut transformer l'observation de colonies de fourmis réelles en algorithme d'optimisation, on reprend l'exemple de ce problème.

Le problème du voyageur de commerce, pour le rappeler brièvement consiste à trouver un chemin Hamiltonien dans un graphe complètement connecté. En termes

plus imagés, il s'agit pour un voyageur de commerce de trouver le chemin le plus court pour visiter une et une seule fois chacune des *n* villes dans lesquelles il doit se rendre. Il s'agit sans doute du problème d'optimisation combinatoire NP-complet le plus utilisé comme test pour les nouvelles méthodes d'optimisation. Voici la modélisation du comportement des fourmis qui est proposée pour le problème du voyageur de commerce.

Les fourmis sont placées sur les sommets du graphe (i.e. sur chaque ville). Elles se déplacent d'un sommet à l'autre en empruntant les arêtes du graphe. Chaque fourmi possède les caractéristiques suivantes :

la fourmi dépose une trace de phéromone sur l'arête (*i, j*) quand elle se déplace de la ville *i* à la ville *j* ;

elle choisit la ville de destination suivant une probabilité qui dépend de la distance entre cette ville et sa position et de la quantité de phéromones présente sur l'arête, (règle de transition) ;

afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville qu'elle a déjà traversée, c'est pour cela que la fourmi doit être dotée d'une mémoire.

Pour éviter qu'une fourmi ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée liste tabou est remise à zéro chaque fois que la fourmi a terminé un tour. La liste tabou constitue la mémoire de la fourmi

Les traces de phéromones sont modélisées par les variables $\tau_{ij}(t)$ qui donnent l'intensité de la trace sur le chemin (*i, j*) à l'instant *t*. La probabilité de transition du sommet *i* vers le sommet *j* par la fourmi *k* est donnée par:

$$pij = \begin{cases} \dfrac{\left|\tau_{ij}(t)\right|^{\alpha} . \left|\nu_{ij}\right|^{\beta}}{\sum_{l \notin L_k(i)} \left|\tau_{il}\right|^{\alpha} \left|\nu_{il}\right|^{\beta}} & Si\ j \notin L_k(i) \\ 0 & \sin on \end{cases}$$

où L_k (*i*) représente la liste tabou de la fourmi *k* située sur le sommet *i* et ν_{ij} représente l'information heuristique, qui correspond à l'inverse de la distance entre les villes *i* et *j*, $\nu_{ij} = 1 / d_{ij}$.

α et β sont deux paramètres permettant de moduler l'importance relative à la phéromone et à l'information heuristique.

Si α = 0, la probabilité sera proportionnelle à $(\nu_{ij})^{\beta}$, les villes les plus proches vont avoir plus de chance à être sélectionnées. Dans ce cas AS va ressembler à un simple algorithme avide et stochastique.

Si β = 0 seule la phéromone sera considérée, cela nous conduira à une émergence rapide de la solution vers un état stagnant.

La mise à jour des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les villes :

$$\tau_{ij} = \tau_{ij}(1-\rho) + \sum_{k=1}^{m} \Delta\tau^{k}{}_{ij}$$

Où ρ est un coefficient représentant l'évaporation des traces de phéromones.
$\Delta\tau^{k}{}_{ij}$ représente la quantité de phéromone déposée sur l'arc (i, j) par la fourmi k :

$$\Delta\tau^{k}{}_{ij} = \begin{cases} 1/L^{k}(t) & \textit{Si l'arc}\,(i,j)\ \textit{est utilisé par la fourmi k} \\ 0 & \textit{Sinon} \end{cases}$$

3.8 Optimisation par essaim particulaire

L'optimisation par essaim particulaire ("Particle Swarm Optimization", PSO) **[KEb95], [EKS 01]**est issue d'une analogie avec les comportements collectifs de déplacements d'animaux. En effet, chez certains groupes d'animaux, comme les bancs de poissons, on peut observer des dynamiques de déplacements relativement complexes, alors que les individus eux-mêmes n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. On peut par exemple observer qu'un banc de poissons est capable d'éviter un prédateur : d'abord en se divisant en deux groupes, puis en reformant le banc originel (figure 3.8), tout en maintenant la cohésion du banc.

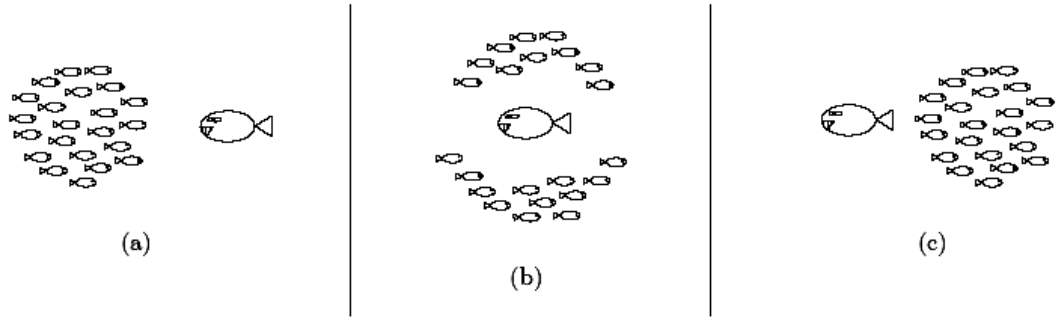


Figure 3.8 Schéma de l'évitement d'un prédateur par un banc de poissons. (a) le banc forme un seul groupe, (b) les individus évitent le prédateur en formant une structure en "fontaine", (c) le banc se reforme.

Ces comportements collectifs s'inscrivent tout à fait dans la théorie de l'auto organisation. Pour résumer, chaque individu utilise l'information locale à laquelle il peut accéder sur le déplacement de ses plus proches voisins pour décider de son propre déplacement. Des règles très simples, comme "rester relativement proche des autres individus", "aller dans la même direction", "à la même vitesse" suffisent à maintenir la cohésion du groupe tout entier, et à permettre des comportements collectifs complexes et adaptés. Les auteurs de la méthode d'optimisation par essaim particulaire se sont inspirés de ces comportements en mettant en perspective la théorie de la socio psychologie sur le traitement de l'information et les prises de décisions dans des groupes sociaux. Cette métaheuristique a été conçue d'emblée dans le cas continu, et c'est toujours dans ce domaine que se situent la majorité des applications à ce jour. La méthode en elle-même met en jeu de larges groupes de particules sous forme de vecteurs se déplaçant sur l'espace de recherche. Chaque particule i est caractérisée par sa position $\vec{x}_i$ et un vecteur de changement de position (appelé vélocité) $\vec{v}_i$ .A chaque itération, la particule se déplace : $\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1)$. Le coeur de la méthode consiste à choisir comment définir $\vec{v}_i$. La socio psychologie suggère que des individus se déplaçant (dans une carte sociocognitive) sont influencés par leur comportement passé et par celui de leurs voisins (voisins dans le réseau social et non nécessairement dans l'espace). On tient donc compte, dans la mise à jour, de la

position des particules, de la direction de leur mouvement (leur vitesse), la meilleure position précédente $\vec{p}_i$ et la meilleure position $\vec{p}_g$ parmi leurs "voisins" :

$$\vec{x}_i(t) = f(\vec{x}_i(t-1), \vec{v}_i(t-1), \vec{p}_i, \vec{p}_g)$$

Le changement de position s'effectue comme suit:

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1(\vec{p}_i - \vec{x}_i(t-1)) + \varphi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1) \end{cases}$$

où les paramètres φ_n sont des variables aléatoires tirées dans U[0; φ_{max}] et qui ont pour rôle de pondérer les rôles relatifs de l'expérience individuelle (φ_1) et de la communication sociale (φ_2).

Le tirage aléatoire uniforme est justifié si l'on ne considère aucun a priori sur l'importance de l'une ou l'autre source d'informations.

3.9 Conclusion

Nous venons de voir les métaheuristiques les plus connues et utilisées. Ces métaheuristiques restent des principes que l'on peut adapter à divers problèmes. Dans le chapitre suivant, nous allons voir deux adaptations des métaheuristiques au problème de la matérialisation des vues.

Chapitre 4

Adaptation des metaheuristiques au problème de la sélection des vues : deux adaptations

Objectif : *l'objectif de ce chapitre est de décrire deux solutions; l'une prend comme contrainte l'espace de stockage, et l'autre travaille sans contrainte. La première solution appelée solution gloutonne part d'un ensemble de solutions vide et le remplit par les vues qui donnent le meilleur bénéfice. Pendant la construction de la solution la méthode gloutonne ne remet pas en cause le choix des vues qui constituent la solution. La contrainte prise en compte (espace de stockage) est moins difficile que la contrainte du coût de maintenance. La seconde solution s'appuie sur la notion de densité et de clusters. Cette méthode travaille aussi avec la notion de bénéfice et ne prend pas en compte de contrainte.*

4.1 Introduction

L'idée de l'entrepôt de données a été introduite pour pouvoir prendre des décisions de manière efficace. Le temps de réponse aux requêtes est l'un des paramètres déterminants dans cette efficacité. Une des techniques utilisées pour réduire ce temps est de sélectionner un ensemble de vues et de les matérialiser. Se pose alors le problème du choix de l'ensemble de vues à matérialiser pour minimiser le temps de réponse globale tout en respectant certaines contraintes liées aux ressources. Les algorithmes exacts de la recherche opérationnelle ne donnent des résultats satisfaisants que pour des tailles réduites du problème. Pour des tailles plus importantes, des heuristiques sont utilisées pour donner des solutions proches de l'optimal. Dans ce qui suit, nous allons présenter deux solutions; l'une sans contraintes et l'autre avec la contrainte espace.

4.2 Solution gloutonne

4.2.1 Introduction

L'algorithme glouton [HAR 96] **(**Greedy Algorithm) utilise une structure du treillis pour représenter les dépendances entre les vues. Les solutions offertes par cet algorithme garantissent un bénéfice d'au moins 63% du bénéfice de l'optimal.

4.2.2 Le model du coût utilisé

Le modèle du coût utilisé est " ***le model du coût linéaire*** ", dans ce modèle le temps de réponse à une requête est égal à l'espace occupé par la vue. Les tailles des vues sont supposées estimées sans les matérialiser.

Le modèle du coût linéaire est défini comme suit : Soit $(L, \preceq)$ un treillis de vues, pour répondre à une requête Q, on choisit un ancêtre de Q, appelé Q_a, déjà matérialisé. On doit traiter la table correspondant à Q_a pour répondre à Q.
Le coût de traitement de Q est donc fonction de la taille de la table associée à Q_a
Il est retenu comme modèle de coût : le coût de traitement de Q est égal au nombre de lignes de la table associée à Q_a utilisés pour construire Q.

4.2.3 L'algorithme glouton (Greedy Algorithme)

On suppose un treillis de vues associé à un cube de données, avec un coût de traitement associé à chaque vue (le coût de traitement est égal au coût d'espace occupé). Soient *c (v)* le coût de traitement de la vue *v*, *k* le nombre de vues à sélectionner et *S* l'ensemble qui va contenir les *k* vues sélectionnées plus la vue sommet qui doit toujours être matérialisée (initialement S = {vue sommet}).
Apres avoir sélectionné un ensemble S de vues, le bénéfice de la vue *v* relatif à *S* noté *B (v,S)*, est défini comme suit :

1. Pour chaque vue $w \leq v$, on définit la quantité B_w par :
    a. Soit *u* la vue qui a le plus petit coût dans *S* tel que $w \leq u$ (*u* est l'ancêtre de *w* qui a le plus petit coût de traitement dans S).
    b. b) Si $c(v) < c(u)$, alors $B_w = c(u) - c(v)$ Sinon $B_w = 0$.
2. On définit le bénéfice de la vue *v* relatif à *S* noté *B (v,S)* par:

$$B(v, S) = \sum_{w \leq v} B_w$$

On calcule le bénéfice d'une vue v en considérant comment cette vue peut améliorer le coût de traitement des vues. Pour chaque vue w que couvre v, on comparons le coût de w en utilisant v et en utilisant l'évaluation la moins chère à partir de S. Si la présence de v améliore, On retient v pour la matérialisation.
L'algorithme glouton pour sélectionner k vues à matérialiser est décrit comme suit :

S = {vue sommet};
Pour *i* = 1 à k faire
DEBUT
Sélectionner la vue v non dans S qui maximise B(v,S)*;*
S = *S* U {*v*};.
FIN ;

S est l'ensemble des vues à matérialiser.
Nous allons prendre un exemple pour illustrer cet algorithme.
Considérons le treillis de la figure 4.1. Ce treillis est constitué de huit vues nommées de *a* à *h*, ayant des coûts d'espace (coût de traitement) indiqués sur le treillis :

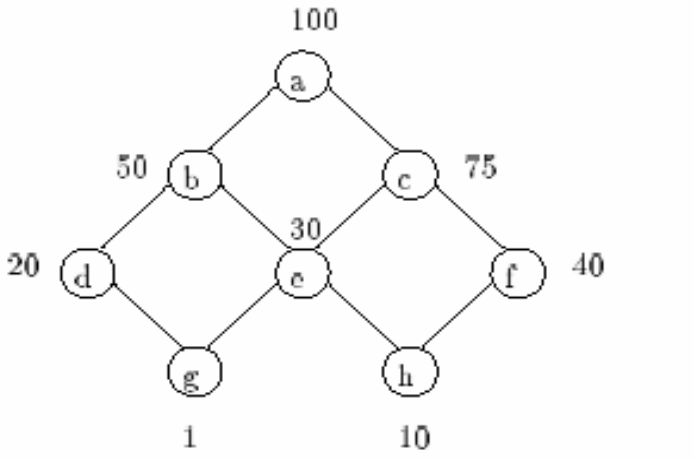


Figure 4.1- Treillis de 8 vues.

La vue sommet *a* doit être choisie pour la matérialisation dès le départ. Supposons que l'on veuille choisir trois autres vues supplémentaires, il faut faire trois choix successifs, donc trois itérations de l'algorithme pour les trouver.

Remarque

Lorsqu'on calcule les bénéfices, on commence avec la supposition que chaque vue est évaluée en utilisant la vue sommet *a*, car initialement S = {*a*}.

Le tableau suivant représente les bénéfices des vues dans chaque itération de l'algorithme.

	1er choix	**2ème choix**	**3ème choix**
b	50* 5= 250		
c	25* 5 =125	25* 2= 50	25* 1 =25
d	80* 2 =160	30* 2= 60	30* 2= 60
e	70* 3 =210	20* 3= 60	20+ 20+ 10 =50
f	60* 2 =120	60 +10= 70	
g	99* 1 =99	49* 1= 49	49* 1= 49
h	90* 1 =90	40* 1= 40	30* 1= 30

Tab. 4.1 - Le bénéfice des vues à chaque itération.

Premier choix : initialement l'ensemble *S* ne contient que la vue sommet *a* donc touts les calculs du bénéfice se font à partir de cette vue. Si on prend la vue *b* pour être matérialisée, on réduit son coût et celui de ses descendants *(d , e, g ,h)* de 50, donc le bénéfice de *B* (*b, S*) = (100-50)*5 = 250. Le même traitement se fait avec toutes les autres vues. Si on prend la vue *e* pour être matérialisée, on réduit son coût ainsi que le coût de ses descendants (*g et h*) de 70 donc le bénéfice de *e* est *B*(*e, S*) = (100-30)*3 = 70*3 = 210.

Pour le premier choix, on trouve que *b* a le plus grand bénéfice *B(b, S)* = 250, *b* est donc considérée comme la première vue matérialisée après *a* (la vue sommet).

S = {*a, b*}.

Deuxième choix : Trouver le deuxième choix consiste à refaire la même opération précédente pour toutes les vues, avec *S* = {*a, b*}.

Par exemple, choisir *f* réduit son coût de 60 (de 100 qui est le coût de *a* à 40 qui est le coût de *f*), la vue *f* a une seule vue descendante *h*. *h* dépend de la vue matérialisée *b* (*b*

est la vue qui a le plus petit coût dans S). Donc B_h = c(u) – c(v) = c(b) – c(f) = 50 – 40 = 10.

Le bénéfice de f est alors B (f, S) = B_f + B_h = 60 + 10 = 70. Comme il est indiqué dans le tableau 4.1 f a le plus grand bénéfice, elle est donc considérée comme la deuxième vue à matérialiser. S = {a, b, f}.

Troisième choix : le troisième choix est indiqué dans la troisième colonne du tableau 4.1. Les mêmes calculs que la première et la deuxième étape se font, mais en considérant l'ensemble S = {a, b, f}. Comme exemple d'un calcul un peut compliqué le calcul du bénéfice de e.

Le choix de e réduit son coût de 20. B_e = c(u) – c(v) = c(b) – c(e) = 50 – 30 = 20 (On a pris b car c'est l'ancêtre matérialisé de e qui a le plus petit coût dans S).

La vue e a deux vues descendantes g et h, leurs bénéfices sont calculés comme suit :

B_h = c(u) – c(v) = c(f) – c(e) = 40 – 30 = 10 (On a pris f car c'est l'ancêtre matérialisé de h qui a le plus petit coût dans S).

B_g = c(u)- c(v) = c(b) – c(e) = 50 – 30, (b est l'ancêtre matérialisé de g qui a le plus petit coût dans S) B_g = 20. Donc le bénéfice de e est :

$$B\ (e,\ S) = \sum_{W \le e} B_w = B_e + B_h + B_g = 20 + 10 + 20 = 50.$$

Après avoir calculé tous les bénéfices des vues restantes, d est sélectionné pour la matérialisation avec un bénéfice égal à 60.

L'ensemble des vues sélectionnées est donc S = {a, b, f, d}.

Coût total = 100 * 8 = 800. (100 est le coût de la vue a, 8 est le nombre de vues)

Bénéfice total = B (b, S) + B (f, S) + B (d, S) = 380.

Cet ensemble réduit le coût de 800 (le cas où seulement la vue a est matérialisée) à 420. (800 – 380 = 420.)

4.2.4 Les performances de l'algorithme glouton

L'exemple suivant illustre un cas où l'algorithme glouton donne le plus mauvais bénéfice. L'algorithme est appliqué sur le treillis ci dessous. La vue sommet de ce treillis est la vue a. Supposons qu'on veuille choisir deux autres vues supplémentaires.

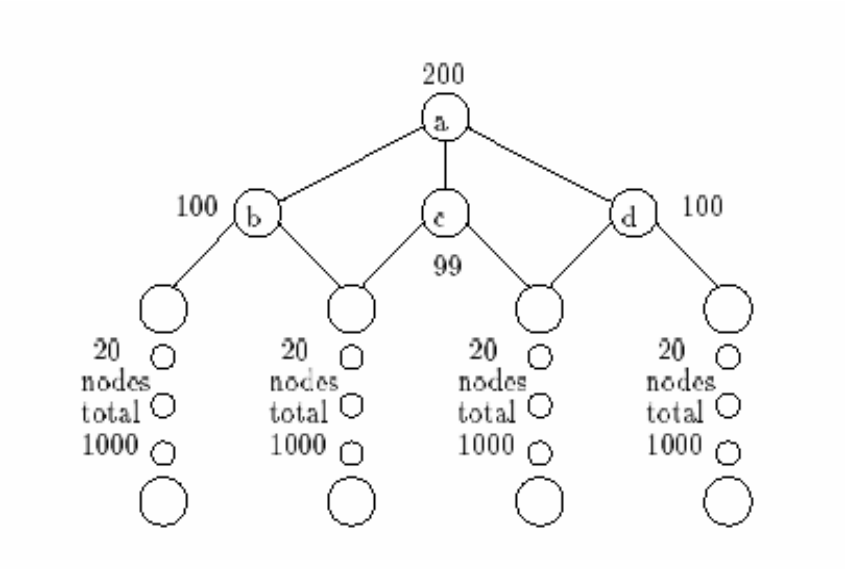


Figure 4.2 - Un mauvais exemple pour la solution gloutonne

1ère choix : D'après l'algorithme glouton, la vue sommet doit être choisie en premier.

Ensuite, on choisi la vue *c* qui a un bénéfice égal à 4141 = (200 - 99) * 41.

Avec : 200 coût de *a.*

99 de *c.*

41 est le nombre de descendants de *c* plus la vue *c* elle-même.

2ème choix : On peut prendre *b* ou *d*, toutes les deux ont un bénéfice égal à 2100 = (200 - 100) * 21.

Avec : 200 coût de *a.*

100 coût de *b.*

21 est le nombre de descendants de *b* plus la vue *b* elle-même.

Alors avec k = 2, l'algorithme glouton produit une solution avec un bénéfice égal à $B_c + B_b = 4141 + 2100 = 6241$.

Cependant, la solution optimale est celle qui prend *b* et *d*. toutes les deux réduisent le coût de 100 ainsi que les 80 vues des quatre chaînes. $B(b, S) = (200-100) * 41 = 4100$. Même bénéfice pour *d.* Donc la solution optimale a un bénéfice de 8200 = 4100 * 2.

Le rapport du bénéfice de l'algorithme glouton / l'algorithme optimal est 6241/8200, qui est 3/4.
Des preuves théoriques dans **[Har 96]** démontrent que le bénéfice de l'algorithme glouton est estimé à au moins 63% du bénéfice de l'algorithme optimal et pas moins.

4.3 Solution basée sur la densité (algorithme DVMA)

4.3.1 Introduction

La matérialisation des vues est une technique bien connue et employée pour accélérer le processus de prise de décision. Dans ce contexte, il y a trois possibilités :

- ***Matérialiser physiquement toutes les vues* :** cette approche donne le meilleur temps de réponse à une requête. Cependant, enregistrer toutes les vues n'est pas une alternative faisable pour les grands cubes de donnes, l'espace consommé devient alors très grand.

- ***Ne rien matérialiser* :** cette approche est la meilleure en espace consommé mais elle est la plus mauvaise en temps de réponse, car On est obligés d'aller à chaque fois aux données brutes pour répondre à une requête.

- ***Matérialiser seulement une partie des vues* :** Dans un cube de données, la valeur de beaucoup de cellules est calculable de celles d'autres cellules. C'est à dire qu'on peut déduire un ensemble de vues à partir d'autres (une vue contient un ensemble de cellules). Au lieu de matérialiser toutes les vues on matérialise un sous ensemble de vues qui permettent de répondre à n'importe quelle requête **[MSR 00].**

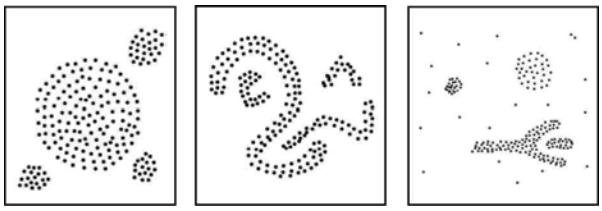
La matérialisation partielle est la solution la plus faisable.
Le problème de la sélection de vues à matérialiser consiste à construire une configuration de vues optimisant le temps de réponse des requêtes. Cette optimisation peut être réalisée sous certaines contraintes telles que l'espace de stockage alloué aux vues sélectionnées ou une borne supérieure du coût de maintenance des vues sélectionnées.

Le problème de la sélection de vues à matérialiser dans les entrepôts de données a été largement étudié tant pour l'approche MOLAP (Multidimentional On-Line Analytical Processing) que pour l'approche ROLAP (Relational On-Line Analytical Processing). Nous avons déjà vu que le problème de sélection de vues matérialisées est NP-complet**.** En effet, si *d* est le nombre de dimensions dans le schéma d'un entrepôt de données, alors le nombre de vues candidates à sélectionner est égal à $n = 2^d$. De nombreux algorithmes ont été développés pour élaborer une solution optimale ou quasi-optimale pour ce problème.

Nous allons présenter dans cette section une approche basée sur la densité (Density-based View Materialization). Cette approche fournit les vues appropriées à être matérialisées dans le but de minimiser le temps de réponse des requête en utilisant une structure le treillis.

4.3.2 Notion de cluster basé sur la densité

Le concept de densité utilisé dans l'algorithme DBSCAN [EKX 96] est basé sur la notion de clusters comme il est illustré dans la figure 4.3 .En regardant les ensembles témoin de points représentés sur cette figure, on peut facilement détecter les points des clusters et les points isolés qui n'appartiennent à aucun de ces clusters



data base 1 **data base 2** **data base 3**

Figure 4.3 ensemble témoin de points

On reconnait les clusters par le fait que dans chaque cluster on a une densité typique de points qui est considérablement plus haute qu'à l'extérieur du cluster. En outre, la densité dans les secteurs des points isolés (noise) est inférieure à la densité dans n'importe quel cluster**.**

Pour formaliser la notion de cluster et de point isolé il faut définir d'abord quelques notions. On va utiliser une base de données *D* de points d'un espace à 2 dimensions S.

L'idée principale est que pour chaque point d'un cluster le voisinage d'un rayon donné doit contenir au moins un nombre minimal de points MinPts, c'est-à-dire que la densité dans le voisinage doit excéder un certain seuil.
La forme d'un voisinage est déterminée par le choix d'une fonction de distance pour deux points *p* et *q*, dénotée par *dist(p,q).*

Définition 4.1 Voisinage d'un point
Le voisinage d'un point, dénoté par $N_{Eps}(p)$ est défini par :

$$N_{Eps}(p) = \{q \in D / dist(p,q) \leq Eps\}$$

une approche intuitive exigerait pour chaque point d'un cluster, qu'il existe au moins un nombre minimum (Minpts) de points dans son voisinage.
Mais cette approche n'est pas très précise car dans un cluster, il existe deux genre de points, les points à l'intérieure du cluster ou point noyau (core point) et les points sur la frontière du cluster ou point frontière (border point)

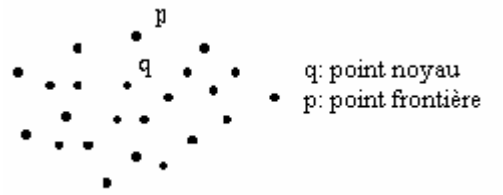


Figure 4.4 : point noyau et point frontière

En général, le voisinage d'un point frontière contient moins de points que le voisinage d'un point noyau. Par conséquent, on doit mettre le nombre minimal de points MinPts à une valeur relativement basse afin d'inclure tous les points appartenant au même cluster.

Définition 4.2 point directement densité-joignable (directly density-reachable): un point p est directement densité-joignable d'un point q si seulement si:

1. $p \in NEps\ (q)$ et
2. $|NEps\ (q)| \geq MinPts$ (la condition d'un point noyau)

Evidement, directement densité-joignable est symétrique pour des pairs de points noyaux, mais pas symétrique si un point est noyau et l'autre un point frontière. La figure 4.4 montre le cas asymétrique.

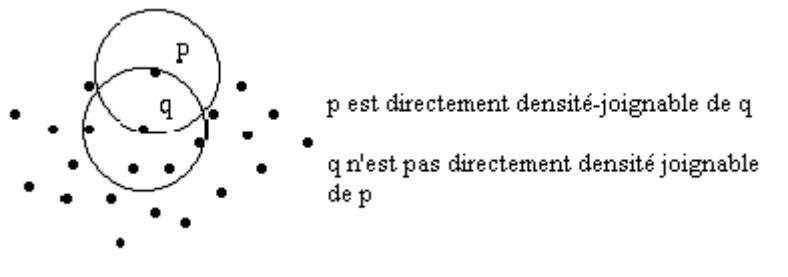


Figure 4.5 point directement densité-joignable

Définition 4.3 point densité-joignable (density-reachable): Un point *p* est densité-joignable d'un point *q* s'il y a une chaîne de points $p_1,.....p_n, p_1 = q, p_n = p$ tel que p_{i+1} est directement densité joignable de p_i.

Cette relation est transitive et non symétrique mais elle est symétrique pour points noyaux , la figure 4.6 montre un cas asymétrique.

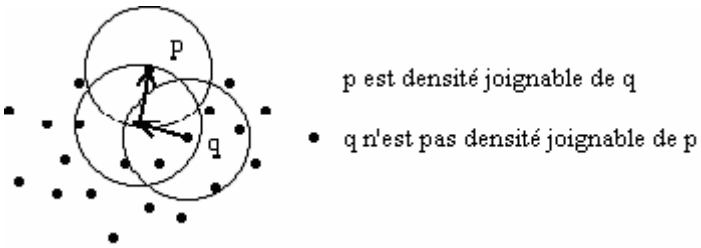


Figure 4.6 point densité-joignable

Deux points frontière du même cluster C ne sont probablement pas densité-joignable entre eux car la condition du point noyau n'est pas verifiée pour les deux. Cependant, il doit y avoir un point noyau dans le cluster dont les deux points frontière du cluster sont densité joignables. Par conséquent, nous présentons la notion de densité-connecté.

Définition 4.4 densité-connecté (density-connected): un point *p* est densité-connecté à un point *q* s'il y a un point *o* tel que les deux points *p* et *q* sont densité-joignable du point *o*. Cette relation est symétrique. La figure 4.7 montre le cas symétrique.

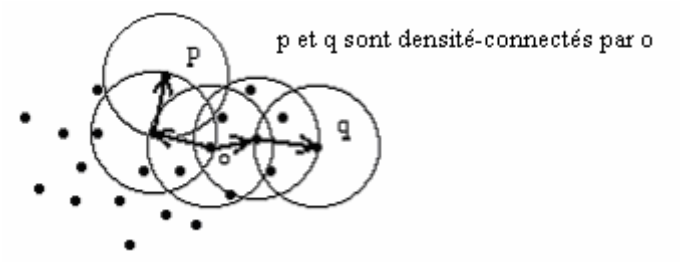


Figure 4.7 densité-connecté

Maintenant, nous pouvons définir la notion de cluster et la notion de point isolé.

Définition 4.5 cluster: Soit D une base de données des points. Un cluster C est un sous- ensemble de D non vide satisfaisant les conditions suivantes:

$\forall p, q$: si $p \in C$ et q est densit-joignable de p, alors $q \in C$.

$\forall p, q \in C$: p est densité-connecté de q.

Définition 4.6 point isolé: étant donnée $C_1, C_2, ..., C_k$ des clusters de la base de données des points D, on définit les points isolés comme un ensemble de points dans D n'appartenant à aucun cluster, c'est-à-dire

$$noise = \{p \in D | \forall i = 1, ..., k : p \notin C_i \}.$$

4.3.3 Concepts utilisés dans l'algorithme DVMA

L'algorithme DBSCAN **[EKX 96]** utilise la notion de densité pour trouver des clusters de formes arbitraire. DVMA utilise également le même concept de densité pour former des clusters de vues et sélectionner les vues à matérialiser dans un entrepôt de données. Un cluster dans l'algorithme DVMA est composé de vues. Le concept principal de DVMA est que le bénéfice du voisinage de n'importe quelle vue dans un cluster doit être au mois égal à une valeur prédéfinie. Le bénéfice représente la qualité d'un cluster. Nous allons définir quelques concepts utilisés dans DVMA.

4.3.3.1 le voisinage d'une vue

Le voisinage d'une vue v noté $N(v)$ est défini par

$$N(v) = v \cup \{w | w \in enfant(v) et |v| - |w| \leq MaxD\}$$

4.3.3.2 la plus proche vue matérialisée parente NMPV (Nearest materialized Parent View)

Cette notion a été introduite dans [URT 99] pour calculer le bénéfice apporté par une vue. On dira qu'une vue u est une vue parente de v si la vue v peut être calculée à partir de la vue u. La vue NMPV (v) est une vue matérialisée u telle que la différence entre la taille de u et de v est minimale parmi toutes les vues matérialisées.

Plus formellement. $NMPV(v) = \min(|v_s| - |v|)) \quad \forall v_s \in S \quad où \quad v_s \xrightarrow{*} v$

Où:

$v_s \xrightarrow{*} v$ Signifie que la vue v_s est l'un des parents de la vue v et S représente l'ensemble des vues à matérialiser.

4.3.3.3 Le bénéfice du voisinage d'une vue

L'algorithme doit trouver des clusters de vues. Pour former ces clusters, il doit calculer le bénéfice du voisinage. Le bénéfice est basé sur la taille des vues, la fréquence d'accès aux vues. La fréquence d'accès aux vues ainsi que la taille des vues peuvent être calculées **[SDNR 96] [LWO 01]**

Le bénéfice du voisinage est donné par:

$$benefit\left(N\left(v\right)\right) = \left(\left|NMPV\left(V\right)\right| - \left|V\right| \sum_{p \in n(v) \cup v} f_p\right) + \sum_{p \in n(v) \cup F} S\left(v\right)$$

où

NMPV(v): la plus proche vue matérialisée parente de la vue v défini précédemment.

f_p : est la fréquence de traitement de la vue p

F est un sous ensemble de vues les plus fréquentes.

$S(v)$ est le support de la vue v.

F et $S(v)$ peuvent être calculées **[DAB 04]**

4.6.3.4 vue noyau

Une vue v est dite vue noyau si seulement si le bénéfice de son voisinage est supérieur où égale à une valeur prédéfinie $MinBen$(minimum de bénéfice), plus formellement :

$$Benefit(N(v)) \geq MinBen$$

4.3.3.5 Notion de directement densité-joignable et densité-joignable

Etant donné un treillis de vues, il existe deux genres de vues : Les vues noyaux et les vues frontière border views (des vues qui n'ont pas de descendants).

Une vue v est directement densité-joignable d'une vue w si seulement si: $v \in N(w)$

w est une vue noyau.

Une vue v est densité-joignable d'une vue w, si seulement si il y a une chaîne de vues $p_1....p_n, p_1 = w, p_n = v$ tel que p_{i+1} est directement densité-joignable de p_i.

Une vue v est densité-connectée à une vue w s'il y a une vue y tel que les deux vues v et w soient densité-joignables à la vue y.

Exemple

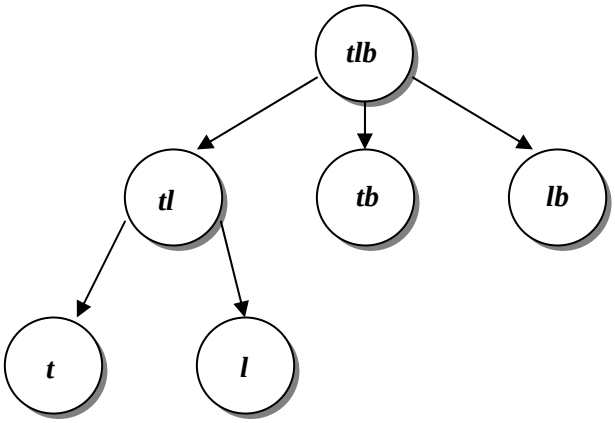


Figure 4.8 : vue noyau vue frontière et vue densité joignable

t, l, tb et ***lb*****:** sont des vues frontières.

tlb et tt : sont des vues noyaux.

tl : est directement densité-joignabe de ***tlb***.

t : est densité-joignable de ***tlb***.

t et ***lb*** sont densité-connecté par ***tlb***.

4.3.3.6 Les catégories de vues

Il existe quatre catégories de vues, les vues classifiées les vues non classifiées, les vues isolées, et les views leaders.

- *Vue classifiée*:c'est une vue traitée et mise dans un cluster.
- *vue non classifiée*: c'est une vue non encore traitée (n'appartient à aucun cluster pour le moment).
- *Vue isolée*: c'est une vue traitée mais qui n'appartient à aucun cluster.
- *Vue leader* : c'est une vue qui n'a pas encore été traitée (non classifiée) mais qui ses ancêtres sont déjà traités (sont classifiés ou isolés).

4.3.3.7 cluster

Etant donné un treillis de vues .Un cluster C est un sous- ensemble de vues non vide satisfaisant les conditions suivantes :

$\forall v, w$: si $v \in C$ et w est densité-joignable de v, alors $w \in C$.

$\forall v, w \in C$: v est densité-connecté à w.

4.3.4 Le principe de l'algorithme DVMA

L'algorithme tente de former des clusters de vues. Pour trouver ces clusters, il calcule le bénéfice des voisinages. L'algorithme travaille sans contraintes et utilise une structure de treillis pour représenter la dépendance entre les vues.

l'algorithme commence à chercher tous les clusters de vues et sélectionne les vues noyaux des clusters pour la matérialisation. Il sélectionne toujours la vue sommet pour la matérialisation, donc on ne prend pas en considération la vue sommet dans la formation des clusters. A chaque itération l'algorithme trouve la plus petite vue leader avec la plus grande dimension car on veut construire des clustrs à partir du haut du treillis. Puis il calcule le bénéfice de N(v). Si ce bénéfice est plus petit que le bénéfice minimum requis (Minben), elle est marquée vue isolée, sinon un cluster démarre de v et tous ses enfants not classifiés sont mis dans une liste de vues candidates. On tire ensuite une des vues candidates et le bénéfice est calculé. Si c'est une vue noyau, tous ses enfants non classifiées sont introduits dans la liste des vues candidates. Si ce n'est pas un une vue noyau, elle est marquée classifiée. Ce processus se poursuit jusqu'à ce

que la liste des vues candidates devienne vide. On a ainsi formé un cluster. De manière similaire on forme les autres clusters. Quand un cluster est formé, la vue noyau du cluster est sélectionnée pour la matérialisation.

L'algorithme DVMA se déroule de la maniére suivante :

- **Initialisation** : l'ensemble des vues à matérialiser contient au début la vue sommet.
- **Sélection d'une vue**: sélectionner la vue leader qui a la taille minimale et la dimension maximale.
- **Création cluster** : numéroter le cluster qui va contenir la vue sélectionnée et toutes les vues qui ont une directement densité-joignable et densité-joignable avec la vue sélectionnée.
- **Evaluation bénéfice du voisinage de la vue** : calculer le bénéfice du voisinage de la vue sélectionnée.
- **Si elle est une vue noyau**: Mettre cette vue dans l'ensemble des vues à matérialiser, et évaluer le bénéfice du voisinage de toutes les vues qui sont directlement densité-joignables et densité-reachables avec cette vue et mettre dans l'ensemble des vues à matérialiser toutes celles qui sont des vues noyaux .
- **Sinon** c'est une vue isolée.
- **Retour à la phase de sélection**.
- **Critère d'arrêt** : la liste des vues leader est vide.

```
Algorithm : DVMA
Input : Lattice of the views, V= Set of views, Access frequency of the
views access, F, MaxD, MinBen
Output: S = Views to be materialized.
Set all views of V as leader.
S = {v1}(S:set of views to be materialized, v1: fact table}
Temp= "True"
clid = Get a new cluster id.
Do while there is a leader view
        Find leader view v with smallest in size (R(v)) among leader
views
           with maximum dimensions.
        Temp= CreateCluster(V, v,clid, MaxD, MinBen)
       If Temp = "True" then
           clid = Get a new cluster id.
       Endif

End DO
```

```
CreateCluster(V, v,clid, MaxD, MinBen )
(Form the cluster with cluster id as clid)
If benefit(N(v)) < MinBen Then
       v.noise= "True"
        Return "False"
Else
        v.classified="True"
       S = S ∪ v
       seeds= {w|w ∈ N(v) and w.classified="False" }
        For all s ∈ seeds set s.classified= "True"
        While Empty(seeds)="False" Do
             For each s ∈ seeds
                If benefit(N(s)) ≥ MinBen then
                    S = S ∪ s
                     Results = {w|w ∈ N(s)}
                     For each r ∈ Results
                        If r.classified = "False" then
                            seeds = seeds ∪ r
                            r.classified="True"
                        Endif
                    EndFor
               Endif
               seeds = seeds − {s}
            Endfor
       EndWhile
       Return "True"
Endif
```

4.4 Conclusion

Apres avoir vu deux solutions pour la résolution du problème de la matérialisation des vues, nous allons voir dans le chapitre suivant l'approche développée pour résoudre ce problème. Cette approche consiste à hybrider deux métaheuristiques (génétique et colonies de fourmis).

Chapitre 5

Approche hybride développée pour la sélection des vues

***Objectif** : l'objectif de ce chapitre est de présenter l'approche développée dans le cadre de cette thèse qui consiste à hybrider un algorithme génétique avec un algorithme des colonies de fourmis. Cette hybridation intervient au moment des croisements et des mutations pour créer de nouvelles populations. L'avantage de ce procédé est l'élimination de l'aspect aveugle de ces deux opérateurs et l' introduction de l'apprentissage. Dans le début du chapitre, nous présentons les outils utilisés ainsi que la formulation du problème. Dans un second temps nous présentons la méthode ainsi que les résultats expérimentaux.*

5.1 Introduction

Un des facteurs importants dans un entrepôt de données est le temps de réponse aux requêtes des décideurs. Le volume important des entrepôts de données entraîne une augmentation de ce temps. D'où la nécessité de techniques d'optimisation pour le réduire. Une des techniques utilisées est la matérialisation des vues. Ainsi lorsqu'une requête est posée, on tentera de répondre à cette requête en utilisant ces vues plutôt que d'utiliser tout l'entrepôt. On pourrait penser à matérialiser toutes les vues mais deux contraintes nous limitent, l'espace de stockage et le coût de maintenance. Comme l'espace de stockage pose de moins en moins de problèmes, L'intérêt devra se porter sur le coût de maintenance. Une vue étant matérialisée, tout changement sur

les données sources devra être répercuté sur les vues matérialisées, ce qui entraîne un coût de maintenance. Ce coût ne doit pas dépasser un certain seuil.
Le but à atteindre est de sélectionner un ensemble de vues à matérialiser afin de minimiser le coût total de traitement des requêtes OLAP en tenant compte d'une contrainte qui peut être l'espace de stockage ou bien le coût de maintenance.
La contrainte d'espace de stockage est la quantité d'espace à ne pas dépasser en matérialisant les vues. La contrainte coût de maintenance est le temps global à ne pas dépasser en maintenant les vues matérialisées.
Dans ce qui suit, nous utilisons une structure de treillis pour exprimer la dépendance qui existe entre les vues. Nous proposons une nouvelle approche qui se base sur une hybridation des algorithmes génétiques avec les colonies de fourmis. Cette hybridation à pour effet d'atténuer l'aspect aveugle des opérateurs génétiques et d'introduire l'aspect apprentissage grâce à la phéromone des fourmis.
Les résultats expérimentaux montrent que l'approche donne de bonnes performances (le rapport de performance avec A* est de 0.90 **[Bou 07])** et génère un nombre satisfaisant de solutions faisables.

5.2 La Modélisation Dimensionnelle

La modélisation dimensionnelle est définie comme une méthode de conception logique souvent associée aux entrepôts de données. Elle consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse.

Chaque modèle dimensionnel se compose d'une table contenant une clé multiple, ***la table de Fait***, et d'un ensemble de tables plus petites nommées ***tables de dimension***. Chaque table « dimension » possède une clé primaire unique qui correspond exactement à l'un des composants de la clé multiple de la table de fait.
Le fait est le sujet à analyser, il est formé de mesures correspondantes aux informations de l'activité analysée

Une dimension se compose de paramètres correspondants aux informations faisant varier les mesures de l'activité. Elle sert à enregistrer les valeurs pour lesquelles sont analysées les mesures de l'activité (fait).

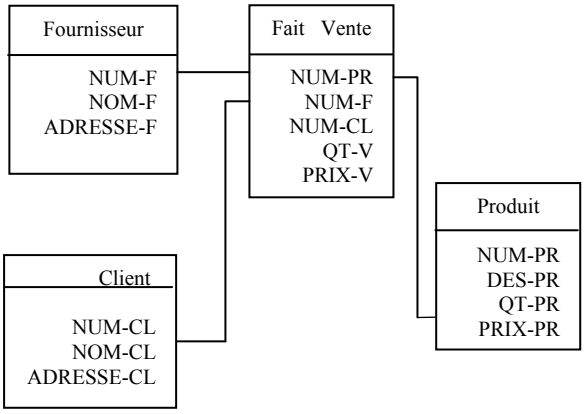


Figure 5.1 Modèle multidimensionnel représentant le fait vente et trois dimensions produit, client et fournisseur

Dans ce modèle des produits sont achetés à des fournisseurs et vendus à des clients à un prix (PRIX-V).

Dans cet exemple nous avons trois dimensions : fournisseur (f) produits (p) et clients (c). La mesure qui nous intéresse est le montant total des ventes. Donc pour chaque point (p,f,c) nous enregistrons le total des ventes pour un produit (p) fourni par un fournisseur (f) et vendu à un client (c). Nous pouvons nous intéresser au total des ventes pour un produit donné (p) pour un client (c) pour tous les fournisseurs**.** **[Gra 96]** introduit une valeur supplémentaire « ALL » au domaine de chaque dimension pour réaliser ceci. La réponse à la requête précédente se trouve au point (p,ALL,c)

5.3 Représentation des relations entre les vues matérialisées.

Nous utilisons une structure de treillis pour représenter les dépendances qui existent entre les vues à matérialiser. En algèbre un treillis est un ensemble ordonné où chaque couple d'éléments admet une borne inférieure et une borne supérieure. Un treillis de vues est un graphe acyclique orienté qui représente la relation hiérarchique qui existe entre toutes les vues. Les nœuds représentent les vues (ou requêtes) et les arcs représentent la relation de dépendance entre les vues, pour deux éléments *x*, *y* de ce graphe il existe un chemin de y vers x si et seulement si la vue *x* dépend de la vue *y*.

Les cellules du treillis sont organisées dans des ensembles différents basés sur la position du ALL dans leur adresse. Par exemple, (p, ALL, c) contient les ventes totales d'un produit p acheté par le client c en considérant tous les fournisseurs. la valeur de la cellule (p, ALL, c) peut être calculée de la somme des valeurs des cellules (p, f_1, c), (p, f_2, c),…, (p, f_n, c), où *n* est le nombre de fournisseurs.

Chacune de ces cellules correspond à une requête SQL différente. Si nous prenons l'ensemble (_, ALL, _) sa valeur est donnée par :

SELECT produit, client, sum PRIX-V as total sales
FROM R
GROUP BY produit, client;

Où **R** représente les données brutes.

Les requêtes qui correspondent aux différents ensembles de cellules diffèrent seulement avec la clause groupe by. Généralement les attributs qui ont la valeur ALL dans la description de l'ensemble des cellules n'apparaissent pas dans la clause group by. Par exemple dans la requête SQL précédente 'fournisseur' a la valeur ALL dans l'ensemble (_, ALL, _) donc il n'apparaît pas dans la clause group by. Puisque les requêtes des ensembles de cellules différent seulement dans le groupement des attributs alors on utilise seulement le groupement des attributs pour identifier les vues **[Har 96]**.

Exemple : revenons à la base de données de la figure 5.1 : nous avons trois dimensions : produit, fournisseur et client représentées respectivement par p, f et c. Nous pouvons les grouper de huit façons différentes :

1) Produit, Fournisseur, Client,
2) Produit, Client,
3) Produit, Fournisseur,
4) Fournisseur, Client,
5) Produit,
6) Fournisseur,
7) Client,
8) Aucun.

Aucun, veut dire qu'il n'y a aucun attribut dans la clause group by, c'est-à-dire qu'elle représente la vue qui contient les ventes totales pour toutes les dimensions.

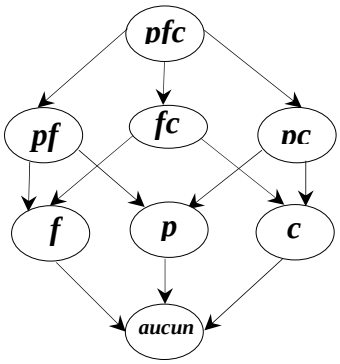


Figure 5.2. représentation des dépendances entre les vues à l'aide d'une structure de treillis

5.4 Formulation du problème

Dans un treillis, les nœuds représentent les requêtes OLAP et les arcs représentent les dépendances entre les requêtes.

Ce treillis est noté $(L, \preceq)$

L : représente l'ensemble des vues.

$\preceq$: représente la relation de dépendance entre les requêtes.

Étant données deux requêtes q_i et q_j, nous dirons que q_i dépend de q_j ($q_i \leq q_j$) si on peut répondre à q_i en utilisant q_j.

Ce treillis peut être représenté par un graphe orienté acyclique G (V, U) où :

V représente l'ensemble des requêtes

V(G) représente l'ensemble des nœuds de G

U(G) représente l'ensemble des arcs de G

Un arc v_i v_j existe dans U si et seulement si $v_j \leq v_i$ et $\nexists$ v_k tel que ($v_j \leq v_k$ et $v_k \leq v_i$) pour $v_i \neq v_k \neq v_j$.

Le graphe G est pondéré de la manière suivante :

Trois poids sont associés à un nœud v :

- r_v : coût de traitement initial de la requête,
- f_v : fréquence d'utilisation de la requête,
- g_v : fréquence de mise à jour de la vue.

Deux poids sont associés aux arcs

- q(u,v) : temps de traitement de la requête u en utilisant v
- m(u,v) : Coût de maintenance de u en utilisant v.

Soit q(u,v) le coût de traitement de la requête u en utilisant v.
q(u,v) est la somme des coûts de traitement associés aux arcs sur le plus court chemin de u à v, plus le coût de traitement initial r_v de v.Si v ne peut pas répondre à u, S_v sera utilisé
m(u,v) est le coût de maintenance qui est la somme des coûts de maintenance du plus court chemin de v à u.
Soit M l'ensemble des vues à matérialiser, q(v,M) dénote le coût minimum de réponse à v en présence des vues matérialisées M et m(v,M) est le coût minimum de maintenance de la vue matérialisée v, en présence de M.
Le problème se formule comme suit
Sélectionner un ensemble M de vues qui minimise T(G,M),
où

$$T(G,M) = \sum_{v \in V} f_v . q(v,M)$$

Sous la contrainte,

$$U(M) = \sum_{v \in M} g_v . m(v,M) \leq S$$

5.5 Algorithmes génétiques adaptés au problème

Une population de génomes est générée aléatoirement représentant la population initiale. Chaque génome représente une solution possible. La longueur du génome est égale au nombre de nœuds du treillis. La valeur 1 dans le génome signifie que le nœud est retenu, la valeur 0 signifie que le nœud n'est pas retenu.

Un génome est représenté comme suit :

Génome=(x_1,…,x_n) où n est le nombre total de nœuds dans le treillis.

$$\begin{cases} x_i = 1 \text{ signifie que la vue } v_i \text{ est matérialisée,} \\ x_i = 0 \text{ signifie que la vue } v_i \text{ nest pas matérialisée} \end{cases}$$

5.5.1 Fonction de pénalité

[Lee 01] utilise un algorithme génétique avec pénalité. Il introduit un coefficient de pénalité r de la manière suivante :

Soit x=(x_1,…,x_n) où x_i = 0 ou 1 et soit $M_x = \{v_i / x_i = 1, i = 1,...n\}$

Le problème de la sélection des vues à matérialiser avec la contrainte coût de maintenance est formulé comme suit:

$$\begin{cases} Max\, f(x) = T(G,\phi) - T(G,M_x) \\ \text{Sous la contrainte} \\ U(M_x) \le S \end{cases}$$

Nous cherchons donc un ensemble de vues M_x qui maximise l'écart entre le temps de traitement global sans aucune matérialisation et le temps de traitement lorsqu'on matérialise M_x.

Soit $\varphi(x)$ la fonction de pénalité

$$\varphi(x) = Max(U(Mx) - S, 0)$$

si une solution x respecte la contrainte alors $\varphi(x)$=0 sinon $\varphi(x)$ est égal au surplus de dépassement de la contrainte.

Le problème initial devient en utilisant la fonction de pénalité.

$$Max\, f(x) = (T(G,\phi) - T(G,M_x)) - r.\varphi(x)$$

où r est le coefficient de pénalité. En augmentant r nous pénalisons plus les solutions infaisables (nous diminuons le nombre de solutions infaisables) et en diminuant r nous autorisons plus de solutions infaisables, (nous rappelons que les solutions infaisables sont admises pour éviter les optimums locaux).

Il n'est pas facile de choisir la bonne valeur du coefficient de pénalité. Etant donné que le choix de r est difficile, une nouvelle technique nommée rangement stochastique est introduite dans **[Jef 03**] pour faire dominer la fonction objectif ou la fonction pénalité.

L'idée est d'introduire une probabilité p_f pour choisir la fonction objectif ou la fonction pénalité.

Durant le processus de rangement on compare deux individus adjacents. Si les deux individus sont des solutions faisables (qui ne violent pas la contrainte), ils seront comparés en utilisant la fonction objectif, cependant si l'une des solutions est infaisable on comparera les deux solutions avec la fonction objectif avec une probabilité p_f et on les comparera avec la fonction pénalité avec une probabilité 1- p_f

Lorsque $p_f > ½$ on avantagera la fonction objectif dans le cas contraire c'est la fonction pénalité qui est avantagée.

On choisira en général $p_f < ½$ pour réduire le pourcentage des solutions infaisables.

5.6 Algorithme de Colonie de fourmis utilisé pour remplacer les opérateurs

Dans le but de rapprocher les algorithmes génétiques des algorithmes à base de colonie de fourmis, nous nous sommes intéressés à un type particulier d'algorithmes évolutionnaires : les algorithmes basés sur la fréquence d'apparition des gènes; au lieu de simuler une population entière de *n* individus qui seront sélectionnés, croisés puis mutés, une *distribution de probabilité* d'apparition des gènes est utilisée pour générer la population. Cette distribution est ensuite modifiée selon l'adaptation des individus de la population.

Un algorithme général nommé BPA (à partir des initiales de BSC, PBIL et ACO) **[Mon 00]** définit la façon générale dont fonctionnent les algorithmes qui se basent sur la fréquence d'apparition des gènes pour résoudre le problème d'optimisation binaire.

Considérons le problème d'optimisation binaire standard qui consiste à trouver dans un espace de recherche $S = \{0, 1\}^l$ le minimum d'une fonction d'évaluation *f*. La valeur du bit *i* de la chaîne *s* sera notée *s* (*i*). Deux structures sont utilisées V et P.

- $V = (p_1, \ldots, p_l)$, avec $pi \in [0, 1]$, représente le vecteur de probabilité qui sera utilisé pour générer des points de S. On décide que p_i représente la probabilité de générer un « 1 »;

- $P = (s_1, \ldots, s_n)$, avec $s_i \in S$, qui représente les n chaînes binaires qui seront générées à chaque cycle. P est considérée comme la population dans les algorithmes évolutionnaires.

Algorithme BPA

(1) Initialisation de $V = (p_1, ..., p_l)$ (en général à (0.5, ..., 0.5))
(2) Tant que La condition d'arrêt n'est pas vérifiée
faire
(3) Générer $P = (s_1, ..., s_n)$ en utilisant V
(4) Evaluer $f(s_1), ..., f(s_n)$
(5) Mettre à jour V selon $(s_1, ..., s_n)$ et $f(s_1), ..., f(s_n)$
(6) fait
(7) Retourner $s+$, la meilleure solution trouvée

La condition d'arrêt peut être de plusieurs types :

- le temps imparti T_1 est atteint ;
- le minimum atteint n'a pas été amélioré depuis T_2 itérations ;
- le nombre d'évaluations de la fonction f a atteint la valeur T_3.

T_1, T_2 ou T_3 sont alors des paramètres d'entrée de BPA.
Nous retenons pour ce qui suit la deuxième condition car nous pensons qu'elle s'adapte mieux au problème.

Les principes d'ACO sont adaptés au problème d'optimisation binaire de la façon suivante : Nous construisons un graphe où chaque sommet correspond à la position d'un bit et où les arcs correspondent au choix de la valeur du bit. Pour construire une solution, la fourmi part du premier sommet sur la gauche et choisit un arc, soit « 1 » ou « 0 », pour atteindre le sommet suivant. La décision de choisir l'arc « 1 » ou l'arc « 0 » suit une distribution de probabilité que l'on appelle trace de phéromones dans

ACO mais qui peut être ramenée à une seule valeur correspondant à la probabilité de suivre l'arc « 1 ».

Notons par 0_i et 1_i les deux arcs correspondant à la position *i* de la chaîne de bits. Les quantités de phéromones de chaque arc sont τ_{0_i} et τ_{1_i}. Ces deux valeurs réelles peuvent être utilisées pour définir une unique valeur p_i, la probabilité de générer un « 1 » :

$$p_i = \frac{\tau_{1_i}}{\tau_{1_i} + \tau_{0_i}}.$$

Initialement, chaque arc 0_i et 1_i ($i \in \{1, \ldots, l\}$) a une quantité de phéromone τ_{0i} et τ_{1i} fixée à une valeur positive τ_0.

5.7 Règle de mise à jour de la phéromone

5.7.1 Mise à jour globale

Dans la règle de mise à jour de la phéromone, nous utiliserons la meilleure chaîne générée depuis le début de l'algorithme (notée s^{++}). Donc la mise à jour de la phéromone, τ_{ki} ($k \in \{0, 1\}, i \in \{1 \ldots l\}$), est faite de la façon suivante:

$$\tau_{ki} = (1 - \rho)\, \tau_{ki} + \rho\, \Delta_{ki}$$

où $\rho \in [0, 1]$ est un paramètre représentant le coefficient d'évaporation des phéromones et Δk_i correspond à la quantité de phéromone déposée sur le meilleur chemin trouvé.

$$\Delta_{k_i} = \begin{cases} \dfrac{1}{1 + c(S^{++})} & if\ S^{++}(i) = k \\ 0 & else \end{cases}$$

où $c(s^{++})$ est le coût de traitement de s^{++}

5.7.2 Mise à jour local

Nous utiliserons une règle de mise à jour locale dans l'étape 3 de la génération de la population de la manière suivante :

$$\tau_{ki} = \tau_{ki} - \alpha\,(\tau_{ki} - \tau_0) \quad \text{si l'arc } k_i \text{ a été choisi}$$

$\alpha \in [0, 1]$ est un paramètre et τ_0 est la quantité de phéromone initiale.

Cette mise à jour locale a pour but de modifier très légèrement la quantité de phéromones sur l'arc choisi par une fourmi afin de pousser les autres à explorer les autres arcs, cela afin d'éviter que toutes les fourmis ne se suivent.

5.7.3 Diversification et intensification

Pour chaque bit généré pour la solution *i* à l'étape 3, nous générons une probabilité *u* :
Si $u < 1 - q_0$
  Alors nous diversifions,
  Sinon nous intensifions.
q_0 est une probabilité donnée au début de l'algorithme

5.8 Résultats expérimentaux

Dans cette section nous présentons quelques résultats expérimentaux relatifs à l'approche proposée. Les programmes ont été écrits en java. La première partie est consacrée à la détermination de quelques paramètres. La seconde partie est consacrée à la comparaison de notre approche avec l'approche proposée dans **[Jef 03]** nommé EA ainsi que l'algorithme A* proposé dans **[Gup 99]**

Programme principal

paramètre : taille de la population p

vecteur de probabilité V
matrice de phéromone τ
et *l* est le nombre de vues.

Début

Initialiser le vecteur V = (p_1, …, p_l) à 0,5.
Initialiser la matrice de la phéromone τ à $\tau_0 = 0,5$
(τ est une matrice 2 * *l*).

// Générer la population initiale G (0)

G (0) ← procédure 1(V, τ);
t=0;
Répéter
t = t + 1 ;
G (t) ← génération d'une nouvelle population
Procédure 1 (V, τ); {Appel de la procédure 1}

S ← classement stochastique (G (t-1) U G (t)) {Appel de la procédure 2}

G (t) ← les P premiers individus de S ;

- Soit s++ la meilleur solution générée depuis le début de l'algorithme, c(s++) son coût de traitement et Δ_{ki} est le taux de phéromone à verser sur les arcs, $\Delta_{ki} = 1/1+c(s++)$.

// Mise à jour de la matrice de phéromone τ.
Pour k = 0 à 1
Pour i = 1 à *l* faire
Si s++ [i] = k **alors** τ [k][i] = $(1 - \rho)$ * τ [k][i] + ρ Δ_{ki} ;
Sinon τ [k][i] = $(1 - \rho)$ * τ [k][i] ;
Finsi
Finpour ;
Finpour ;

// Mise à jour du vecteur V
Pour i = 1 à *l* faire
V[i] = τ [1][i] / τ [1][i] + τ [0][i] ;
Finpour ;

Jusqu'à (la condition d'arrêt)
Fin

Procédure 1
Paramètre : q_0 , α ; $k \in \{0,1\}$// k représente l'arc «0» ou l'arc «1».
Début
Pour j = 1 à *n* // n st le nombre d'individus.
Pour i = 1 à *l*
Générer une probabilité $u \in [0, 1]$
Si $u < (1 - q_0)$ // diversifier
Générer une probabilité $w \in [0, 1]$
Si $w <$ *pi*
Le bit *i* de la chaîne Sj prend la valeur « 1 » donc k =1.
Sinon
Le bit *i* de la chaîne Sj prend la valeur « 0 » donc k = 0.
Finsi
// Mise à jour local
τ [k][i] $\leftarrow (1 - \alpha) * \tau$ [k][i] + $\alpha \tau_0$;

Sinon // intensifier
Si (pi > 0,5) **alors**
Le bit i de la chaîne Sj prend la valeur « 1 » donc k =1 ;

Sinon il prend la valeur « 0 » k = 0 ;
// Mise à jour local
τ [k][i] $\leftarrow (1 - \alpha) * \tau$ [k][i] + $\alpha \tau_0$;
Finsi

Finsi
Finpour ;
Finpour ;
Fin.

Procédure 2

Entrée : λ = 2*P individus { Ij \ j=1,…, λ}
Paramètre : paramètre d'équilibre Pf
Début
Pour i=1 à N **faire**
Pour j=1 à λ-1 **faire**
Prendre u $\in$ *U (0,1)* ;
// u est nombre aléatoire choisie entre 0 et 1.

Si (Ø(Ij)= Ø(Ij+1) = 0) ou (u < Pf) **alors**

Si (f(Ij) <f (Ij+1)) **alors**
Échanger (Ij, Ij+1) ;
Fin si
Sinon

si Ø(Ij) > Ø(Ij+1) **alors**
Échanger (Ij, Ij+1) ;
Fin si ;
Fin si ;
Fin pour ;
Si il n'y a pas eu d'échange **alors**
Quitter ;
Fin si ;
Fin pour ;
Fin.

5.8.1 Resultats experimentaux relatifs à l'algorithme proposé

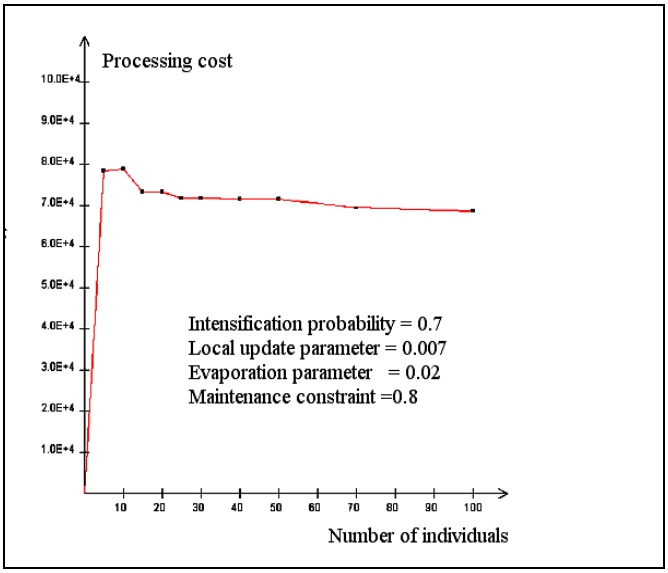


Figure 5.3 Coût de traitement en fonction du nombre d'individus

Nous remarquons que le coût de traitement diminue en augmentant le nombre d'individus de la population. Le coût de traitement se stabilise à environ 7.10^4 pour 60 individus

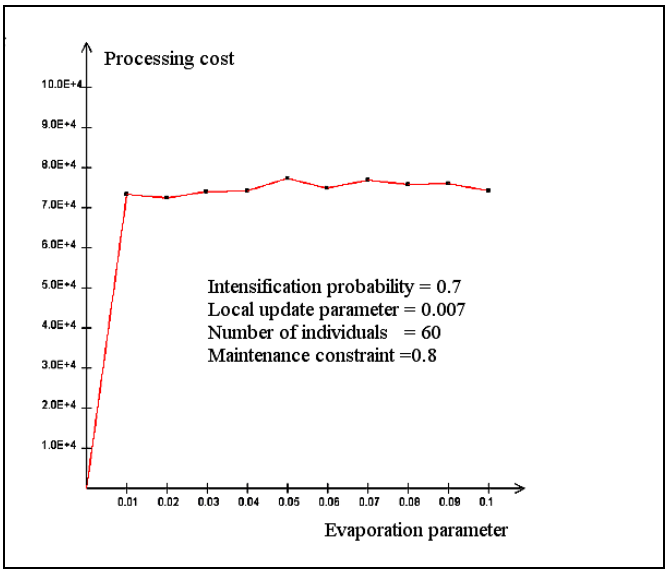


Figure 5.4 Coût de traitement en fonction du paramètre d'évaporation

Nous remarquons que le coût de traitement est minimum lorsque le paramètre d'évaporation est égal à 0.02

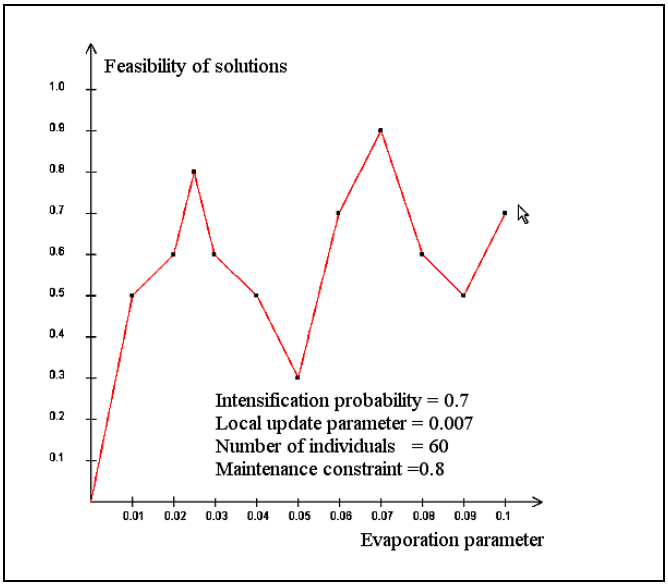


Figure 5.5 Graphe faisabilité des solutions en fonction du paramètre d'évaporation

Nous remarquons que les deux meilleures solutions pour la faisabilité sont pour le taux d'évaporation égale à 0.025 et 0.07 mais d'après le graphe précédant le coût de traitement est meilleur pour 0.025. Nous retenons donc pour le paramètre d'évaporation la valeur 0.025.

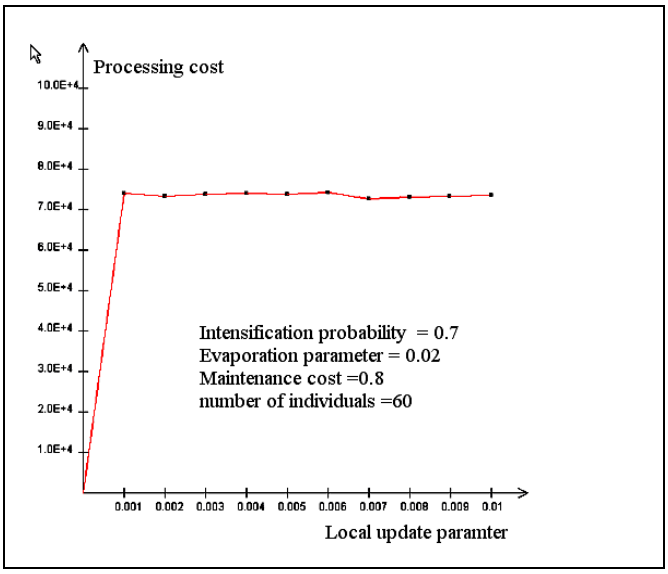


Figure 5.6 Coût de traitement en fonction du paramètre de mise à jour locale

Ce graphe montre que le coût de traitement est meilleur pour le paramètre de mise à jour locale 0.007.

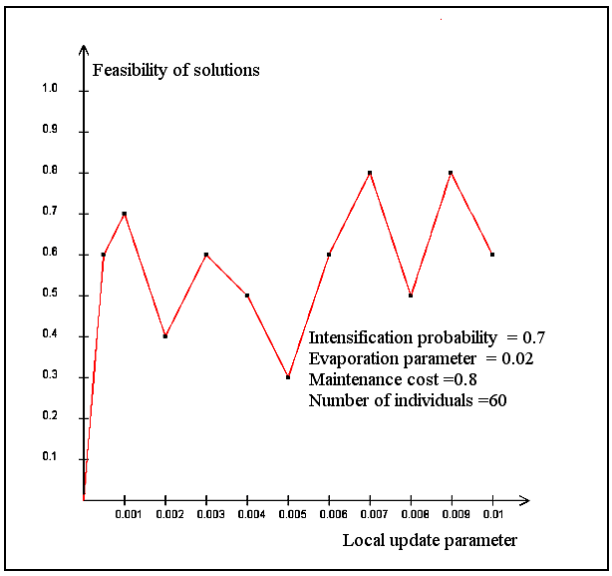


Figure 5.7 Faisabilité des solutions en fonction du paramètre de mise à jour locale

Ce graphe montre que le taux de solutions faisables est meilleur pour les deux valeurs 0.007 et 0.009 mais le coût de traitement est meilleur pour 0.007 donc nous fixons le paramètre de mise à jour locale à 0.007

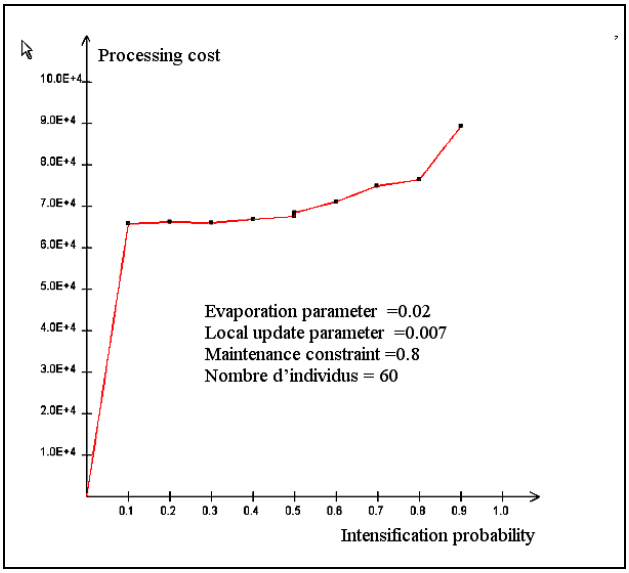


Figure 5.8 coût de traitement en fonction de la probabilité d'intensification

Nous remarquons que le coût de traitement augmente légèrement entre 0.5 et 0.8 puis il augmente rapidement.

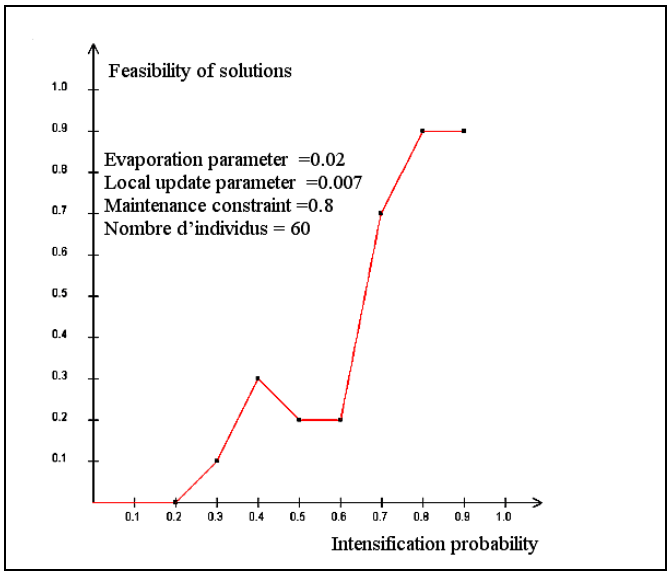


Figure 5.9 Faisabilité des solutions en fonction de la probabilité d'intensification

Dans ce graphe le pourcentage des solutions faisables est raisonnable à partir de 0.7 et comme le coût de traitement est très grand à partir de 0.8 alors nous fixons la probabilité d'intensification à 0.7.

5.8.2 Comparaison de l'algorithme proposé avec l'algorithme A*

Nous estimons le coût de traitement à partir d'un coût de maintenance egal à 0.4 car pour une contrainte inférieure à 0.4, l'algorithme hybride donne un pourcentage élevé de solutions infaisables.

Table 5.1.Comparaison entre l'algorithme hybride et A* pour différents coûts de maintenance

Coût de la méthode		**Seuil**
CH	**CA***	**S**
73266	69632	0.4
72456	66216	0.5
70218	65408	0.6
75928	64979	0.7
74381	64979	0.8
70408	64979	0.9
74828	64588	1

CH: Coût de traitement de l'algorithme hybride,
CA*: Coût de traitement de l'algorithme A* ,
S: Seuil du coût de maintenance exprimé en fraction du coût de maintenance global.
Le rapport de performance p est

$$p = \frac{\sum_{i=1}^{n} \frac{CA_i}{CH_i}}{n} = 0.90$$

où n représente le nombre d'instances.

Nous notons que le rapport de performance est supérieur à 4/5, Ce qui donne une bonne performance à notre algorithme.

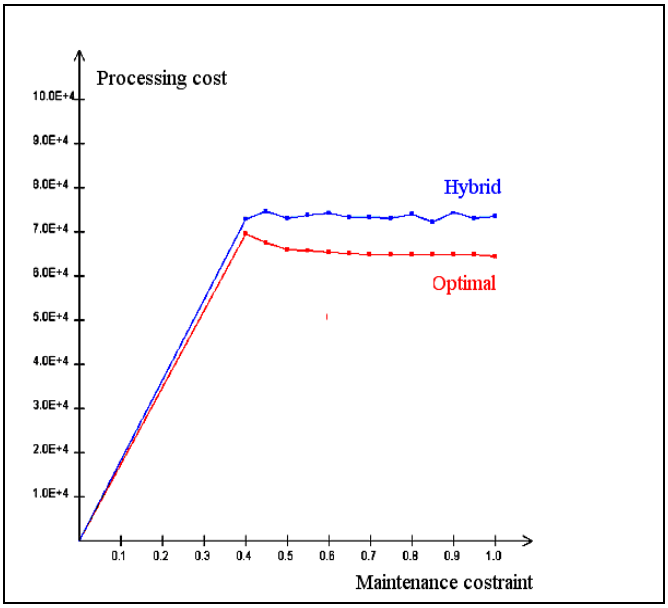


Figure. 5.10 coût de traitement en fonction de la contrainte de maintenance

5.8.3 Comparaison de l'algorithme proposé avec l'algorithme EA de [Jef 03]

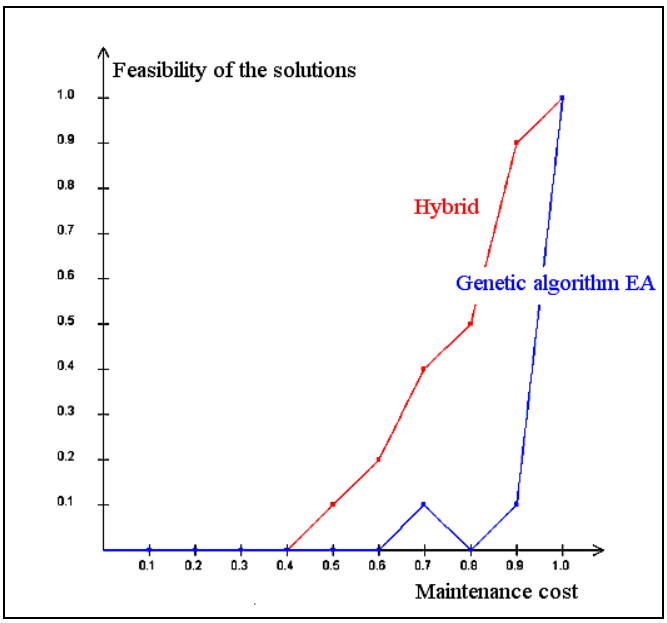


Figure 5.11 Faisabilité des solutions en fonction de la contrainte de maintenance

Nous remarquons que le nombre de solutions faisables issues de l'algorithme hybride est supérieur à celui de l'algorithme génétique de [Jef 03]. Nous pensons que ceci est du au fait que l'algorithme hybride est guidé par l'expérience des fourmis dans la génération des populations, par contre l'algorithme génétique génère les populations de manière aveugle (croisement et mutation).

.

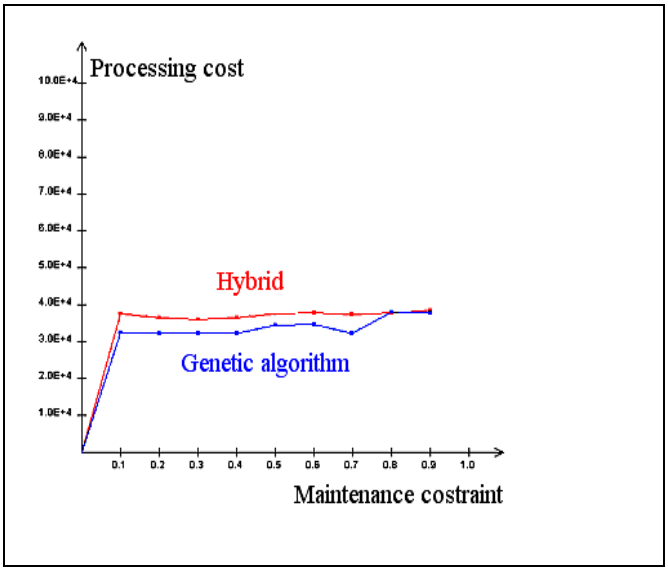


Figure 5.12 Coût de traitement en fonction de la contrainte de maintenance

Nous remarquons dans ce graphe que le coût de traitement de l'algorithme hybride est légèrement plus élevé par rapport à celui de l'algorithme génétique de **[Jef 03]**, ceci est du au grand nombre de solutions infaisables générées par l'algorithme génétique.

Table5.2*Comparaison entre EA et l'algorithme hybride pour différentes dimensions*

dimension	**02 (04 Noeuds)**		**03 (08 Noeuds)**	
Algo.	**EA**	**Hybrid**	**EA**	**Hybride**
M.P.C.	1208400	1208400	$5.28\ 10^7$	$5.28\ 10^7$
P.C.S.	4916	2458	62911.20	28635.67
Benef	1203484	1205942	52737089	52771365
Temps	0.67	1.75	2.63	5.38

dimension	**04 (16 Noeuds)**		**05 (32 Noeuds)**	
Algo.	**EA**	**Hybrid**	**EA**	**Hybrid**
M.P.C.	2046000	2046000	8938500	8938500
P.C.S.	144881	65146	73578	22095
Benef.	1901119	1980854	8864922	8916405
Temps	12.43	15.65	95.18	97.45

dimension	**06 (64 Noeuds)**		**07 (128 Noeuds)**	
Alg	**EA**	**Hybrid**	**EA**	**Hybrid**
M.P.C.	$2.68\ 10^7$	$2.68\ 10^7$	$9.37\ 10^7$	$9.37\ 10^7$
P.C.S.	127980	279680	5045438.8	1036912.4
Benef.	26672020	26520320	88654562	92663088
Temps	752.14	303.10	4935.07	947.97

MPC: Coût de traitement maximum (Coût de traitement sans matérialisation)
PCS: Coût de traitement de la solution
Benef: Benefice de la solution (Benef = MPC - PCS),
Time: Temps d'exécution en secondes.

Dans cette table, nous remarquons que l'algorithme EA est plus rapide que l'algorithme hybride pour les petites dimensions, mais que pour de plus grandes dimensions l'algorithme hybride est plus rapide. Nous remarquons aussi que l'algorithme hybride est scalable. Ceci est plus visible sur la représentation graphique de la figure 5.13

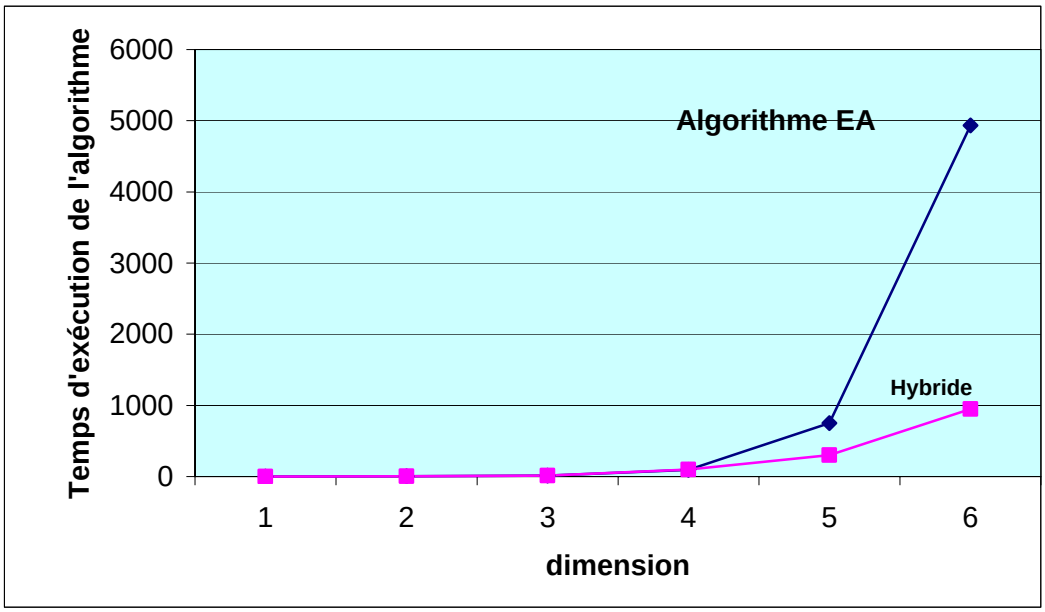


Figure 5.13 Evolution du temps d'exécution en fonction des dimensions

Nous remarquons sur ce graphe que le temps d'exécution dans l'approche hybride suit une allure meilleure que celle de l'algorithme génétique EA.

5.9 Conclusion

Nous venons de voir l'approche proposée ainsi que la validation expérimentale des performances de cette approche qui peut être résumée dans ce qui suit :

- Bon rapport de performance avec une méthode exacte
- Meilleur temps d'exécution que EA pour de grandes dimensions
- Meilleur nombre de solutions faisables.
- Scalabilite.

Conclusion générale

Un entrepôt de données est construit en intégrant plusieurs sources de données éventuellement hétérogènes dans le but de l'analyse et de la prise de décision. Dans ce domaine, plusieurs axes de recherches sont actuellement ouverts. Nous pouvons citer l'intégration des données et des schémas, la modélisation dimensionnelle, la fouille des données, le stockage efficace des données, la maintenance etc.…

Le temps de réponse dans un entrepôt de données est l'un des paramètres les plus importants et ce pour deux raisons principales :

- Le temps dans la prise de décision est vital dans une entreprise
- Le volume de l'entrepôt augmente continûment

Plusieurs solutions ont été proposées pour optimiser ce temps. Parmi ces solutions, nous trouvons la matérialisation des vues. Cette technique consiste à précalculer certaines requêtes et à les stocker physiquement évitant ainsi l'accès à tout l'entrepôt. Cette technique induit deux problèmes : La sélection des vues à matérialiser et la maintenance des vues matérialisées.

Dans cette thèse, nous avons abordé le problème de la sélection des vues à matérialiser

Il existe trois possibilités pour sélectionner un ensemble de vues à matérialiser :

- Matérialiser toutes les vues : Cette approche donne le meilleur temps de réponse, mais présente l'inconvénient du coût de stockage et de maintenance élevés
- Ne matérialiser aucune vue : Cette approche ne coûte rien en stockage ni en maintenance, mais ne fourni aucun avantage pour les performances des requêtes.
- Matérialiser seulement une partie des vues : Dans cette approche, on sélectionne un ensemble de vues à matérialiser.

Dans le cadre de cette thèse, nous avons opté pour la 3^{eme} approche. Il a été montré que le problème à résoudre dans cette approche est NP-difficile [Gup 99]. Les méthodes exactes ne donnent des résultats satisfaisants que pour des tailles réduites du problème. L'utilisation des métaheuristiques s'impose. Nous avons proposé dans le cadre de cette thèse, un algorithme hybride qui utilise les algorithmes génétiques et les colonies de fourmis. Le mariage de ces deux metaheuristiques tente de supprimer l'aspect aveugle des opérateurs des algorithmes génétiques en introduisant l'apprentissage induit par la phéromone des fourmis.

L'étude expérimentale a été menée en trois phases :

1. fixer les paramètres de notre algorithme hybride
2. comparaison de l'algorithme hybride avec un algorithme exact et calcul du rapport de performance
3. comparaison de l'algorithme hybride avec un algorithme évolutionnaire performant nommé EA [Jef 03]
4. Etude de la scalabilité de l'algorithme hybride

Les résultats de cette étude ont montré ce qui suit :

- Le rapport de performance de l'algorithme hybride par rapport à l'algorithme exact est égal à 0.90. Ce qui est une bonne performance
- Le nombre de solutions faisables de l'algorithme hybride est plus élevé que celui de EA
- Pour de petites dimensions, le temps d'exécution de EA est meilleur que l'algorithme hybride, mais pour de grandes dimensions, l'hybride est meilleur

Perspectives

Le travail réalisé dans cette thèse ouvre diverses perspectives. Nous envisageons de prospecter la voie de la concurrence entre plusieurs metaheuristiques. Dans cette approche, plusieurs metaheristiques vont travailler en compétition dans l'espace des solutions, elles seront supervisées par une autre metaheuristique de niveau supérieur (hyperheuristique). La voie du clustering semble aussi prometteuse. Nous envisageons aussi de prospecter l'aspect biobjectif où le temps et l'espace seront deux objectifs contradictoires à satisfaire et le coût de maintenance une contrainte.

Bibliographie

[Agr 96] R. Agrawal, A. Gupta And S. Sarawagi Modelling multidimensional databases. Research report: IBM *Almaden research center, San jose, CA, 1996*

[AMS 03] S. Aupetit, N. Monmarché, M. Slimane, C; Guinot and G. Venturini. Clustering and Dynamic Data Visualization with Artificial Flying Insect. In Genetic and Evolutionary Computation Conference (GECCO). 2003

[Ark 98] R. Arkin Behavior-bsed robotics, Mit press, Cambridge, Massachusetts 1998

[BDT 99] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence, From Natural to Artificial Systems. Oxford University Press. 1999

[Bla 86] J. A. Blakeley, P. Larson anf F.W. Tompa. Efficiently updating materialized views, Proceeding of the ACM SIGMOD international conference on management of data, Page 61-71. 1986

[Bou 06] A. Boukra, M. Ahmed-Nacer and S. Bouroubi, Selection of views to materialize in data warehouse. In EURO XXI Conference Reykjavik Iceland 2006

[Bou 07] A. Boukra, M. Ahmed-Nacer and S. Bouroubi, Selection of views to materialize in data warehouse A hybride solution in international journal of computational intelligence research (IJCIR) issue 4, volume 3, 2007

[CDF 00] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. Self-Organization in Biological Systems. Princeton University Press. 2000.

[Cho 02] C. -H. Choi, J. X. Yu, and G. Gou, What difference heuristics make : Maintenance-cost view-selection revisited in Proc. Third Int. Conf. Web-Age Information Management, 2002.

[CiS 01] V. Cicirello and S. Smith Wasp-like Agents for distributed Factory Coordination. Technical Report CMU-RI-TR-01-39, Robotics Institute, Carnegie Mellon University, Pittsburgh. 2001.

[DAB 04] Das and D K Bhattacharyya. Faster Algorithms for Association Rule Mining.In Proceedings of 12th International Conference on Advanced Computing and communication, Ahmedabad, India, Dec, 2004.

[DeV 99] L. De Castro and F. Von Zuben. Artificial Immune Systems : Part I : Basic Theory and Applications. Technical Report TR-DCA 01/99, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil. 1999

[Dor 01] M. Dorigo. The ant colony optimization metaheuristic : algorithms, application, and advances », Technical Report IRIDIA-2000-32, IRIDIA, Université Libre deBruxelles, Belgium, 2001.

[EKX 96] M Ester,H P Krigel,J Sander and X Xu . A density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining, Portland, , pp 226-231. 1996

[Fer 97] J. Ferber. Les systémes multi-agents. Vers une intelligence collective. InterEditions. 1997.

[Fra 01] J.M. Franco S. Delignelles Piloter l'entreprise grâce au data warehouse édition Eyrolles 2001

[Gar 00] G. Gardarin. Internet/intranet et base de données , Edition Eyrolles 2000

[Glo 89] F. Glover, Tabou search – part I. ORSA Journal on computing, tome 1 190-206, 1989

[Gol 94] D. E. Goldberg. Algorithmes génétiques. Exploration, optimisation et apprentissage automatique. Addison-Wesley France. 1994

[Gra 96] J. Gray, A. Bosworth, A. Layman and H. Pirahesh. Datacube : a relational Agregation Operateur Generalizing Group By, Cross-tab, and Sub-totals , IEEE Int. Conf. On Data Enegineering, pp.152-159, 1996.

[Gue 01] E.B.Guerrero, C. Collet, and M. Adiba. Entrepôts de données: caractéristiques et problématique. Technique et science informatiques, 145 – 178, 2001.

[Gup 97] H. Gupta,V. Harinarayan, A. Rajaraman, and J. D. Ullman, Index selection for OLAP, in Proc. Thirteenth Int. Conf. Data Engineering, 1997.

[Gup 99] H. Gupta and I. S. Mumick, Selection of views to materialize under a maintenance cost constraint, in Proc. 7th Int. Conf. Database Theory, pp. 453–470. 1999

[Har 96] V. Harinarayan, A. Rajaraman, and J. D. Ullman, Implementing data cubes efficiently, in Proc. 1996 ACM SIGMOD Int. Conf. Management of Data, , pp. 205–216. 1996.

[HGH99] J. K Hao, P. Galinier, et M. Habib. Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes , Revue d'Intelligence Artificielle., vol : no.1 1999.

[Hol 92] J.H. Holland Adaptation in natural and artificial systems 2nd edition MIT press 1992

[Jef 03] Jeffrey Xu Yu, Xin Yao, Chi-Hon Choi, and Gang Gou Materialized View Selection as Constrained Evolutionary Optimization IEEE transactions, man and cybernetics. Part c:applications and reviews vol 33, NO 4, November 2003

[Jen 96] N.R. Jennings,. Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings, editor, Foundations of Distributed Arti_cial Intelligence, pages 187{210. John Wiley & Sons. 1996

[KEb 95] J. Kennedy, and R. C. Eberhart,. Particle swarm optimization. In Proc. IEEE Int. Conf. on Neural Networks, volume IV, pages 1942 -1948, Piscataway, NJ : IEEE Service Center. 1995

[Kim 96] R. Kimball The data Warehouse toolkit John Wiley an sons 1996

[kim 97] R. Kimball. A dimensional modeling manifesto. DBMS Magazine, August 1997

[Kir 83] S. Kirkpatrick, C. Gelatt, and M. vecchi, Optimization by simulated annealing. Science, tome 220, n° 4598, Pages 671-680,1983

[KoR 86] Dynamat : A dynamic view mangement system for Data warehouse. Proceedings of the ACM SIGMOD international Conference on management of data pages 371-382 june 1986

[Lee 01] M. Lee and J. Hammer, Speeding up materialized view selection in data warehouses using a randomized algorithm, Int. J. Cooperative Inform. Syst., vol 10 no.3, pp 327-353, 2001

[Lin 86] B.G. Lindsay, L. M. Haas, C. Mohan, H. Pirahesh and P.F. Wilms. Asnapshot, differential refresh algorithm. Proceeding of the ACM SIGMOD international conference of mnagement of data base, page 53-60 June 1986.

[LWO 01] W.Liang,H Wang and M E Orlowska. Materialized View Selection Under Time Constraint. Data and Knowledge Engineering, pp203-216, 2001.

[Mar 04] M.Sevaux. Métaheuristiques Stratégies pour l' optimisation de la production de biens et de services. Thèse d'habilitation Préparée au Laboratoire d'Automatique, de Mécanique d'informatique Industrielles et Humaines du CNRS (UMR CNRS 8530) dans l´équipe Systèmes de Production. Juillet 2004

[Mon 00] N. Monmarché. Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation ». Thèse de doctorat. Ecole Doctorale : Santé, Sciences et Technologies. Décembre 2000.

[MSR 00] M. Mohania, S.Samtani, J.F.Roddick, and Y. Kambayashi. Advances and research directions in data warehousing technology. in the Australian Journal of Information Systems, 2000.

[PrG 71] I. Prigogine and P. Glandsdorf. Thermodynamic Theory and Structure, Stability and Fluctuations. Wiley and Sons, New York. 1971.

[ROU 01] O.Roux . La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation et à la programmation automatique . Thèse de doctorat. Université du littoral Côte d'Opale. Décembre 2001.

[SDNR 96] A.Shukla, P.M. Deshpande, J.F.Naughton, K.Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies _,in 22nd International Conference on Very Large Data Bases (VLDB 1996), Bombay, India, pp. 522_531. 1996.

[SEG 03] M. Segond. Application d'un algorithme à colonie de fourmis à la détection de structures retentives en eaux côtières , Laboratoire d'Informatique du Littoral, Maison de la Recherche Blaise Pascal. France, 2003.

[Shu 98] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. in Proc. 24th Int. Conf. Very Large Data Bases, pp. 488–499. 1998

[Tes 00] O. Teste Modelisation et manipulation d'entrepôts de données complexes et historisés. Thèses de doctorat. Institut de recherche en informatique de Toulouse. Publication, laboratoire IRIT Pole Sig 2000

[URT 99] H. Uchiyama, K. Runapongsa, T.J. Teorey. A Progressive View Materialization Algorithm , in 2nd ACM International Workshop on Data warehousing and OLAP (DOLAP 1999), Kansas City, USA, pp. 36_41. 1999.

[Vas 99] P. Vassiliadis and T. Sellis. A survey of logical models for olap databases. SIGMOD *record,* 28(4):64-69, December 1999

[YKL97] J. Yang, K. Karlapalem et Q. Li Algorithms for materialized views design in data warehouseing environment. Proceedings of the international conference on very large Data bases, pages 136-145 august 1997.

[Zha 01] C. Zhang, X. Yao, and J. Yang, An evolutionary approach to materialized view selection in a data warehouse environment, IEEE Trans. Syst., Man, Cybern. C, vol. 31, pp. 282–294, Aug. 2001.