كلية العلوم و التقنيات فاس
ⵜⴰⵖⵣⵓⵍⵜ ⵏ ⵜⵓⵙⵙⵓⵔⵉⵏ ⴷ ⵜⵙⵏⵙⵜⵉⵏ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴰⴱⴷⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah

N° d'ordre 54/2018

THESE DE DOCTORAT

Présentée par
Mohammed EL ALAOUI

Spécialité : **Recherche opérationnelle et Informatique**

Sujet de thèse

Optimisation des requêtes dans les bases de données décisionnelles via la programmation par contraintes.

Thèse présentée et soutenue le vendredi 16 novembre 2018 à 15h devant le jury composé de :

Nom	**Titre**	**Etablissement**	
Abdelmajid HILALI	PES	Faculté de sciences et techniques de Fès	Président
Mohamed SABBANE	PES	Faculté des sciences de Meknès	Rapporteur
Abdellatif El AFIA	PES	ENSIAS de Rabat	Rapporteur
Ahmed EL HILALI ALAOUI	PES	Faculté de sciences et techniques de Fès	Rapporteur
Hamid TAIRI	PES	Faculté des Sciences Dhar El Mahraz de Fes	Examinateur
Khalid HADDOUCH	PH	ENSA de Al-Hoceima	Examinateur
Mohamed ETTAOUIL	PES	Faculté de sciences et techniques de Fès	Directeur de thèse

Laboratoire d'accueil : Modélisation et calcul scientifique.

Etablissement : Faculté des Sciences et Techniques de Fès.

Remerciement

Le travail présenté dans ce rapport de thèse est le fruit de la collaboration de plusieurs chercheurs que je tiens à les remercier vivement de m'avoir accompagné, soutenu et guidé tout au long de mon cursus.

Je voudrais tout d'abord exprimer tous mes respects et remerciements à mon directeur de thèse, Monsieur Mohamed ETTAOUIL, Professeur à la Faculté des Sciences et Techniques de Fès, pour sa disponibilité, sa grande patience, ses précieux conseils et ses encouragements m'ont permis de préparer ma thèse dans des bonnes conditions. Enfin, j'ai été extrêmement sensible à ses précieuses qualités humaines et son comportement de père plus que professeur qui m'ont offerts la chance de travailler dans un environnement familial.

Mes remerciements les plus profonds sont adressés à Monsieur HILALI Abdelmajid, Professeur à la Faculté des Sciences et Techniques de Fès, d'avoir accepté de présidé le jury de ma thèse malgré ses nombreuses occupations.

J'exprime ma profonde gratitude à Monsieur Mohamed SABBANE, Professeur à la Faculté des Sciences de Meknès, pour le temps précieux qu'il m'a accordé en acceptant d'être rapporteur de ce travail malgré ses nombreuses occupations. Je le remercie pour toute attention qu'il m'a accordée, et cela malgré son temps très chargé.

Je tiens à remercier également Monsieur ELAFIA Abdelatif, Professeur à l'ENSIAS de Rabat, de m'avoir fait le privilège de rapporter mon manuscrit et de se déplacer de loin pour juger mon travail de thèse. Je le remercie également pour ses précieux conseils.

Mes remerciements les plus respectueux s'adressent à Monsieur ELHILALI Alaoui Ahmed, Professeur à la Faculté des Sciences et Techniques de Fès, pour l'honneur qu'il m'a fait en acceptant d'être le rapporteur de ce travail. Je ne raterai pas cette occasion sans le remercier pour ses encouragements, ses compliments et son accompagnement durant mes années d'études à la faculté de sciences et techniques.

J'exprime ma profonde gratitude à TAIRI Hamid, Professeur à la Faculté des Sciences de Fès, pour le temps précieux qu'il m'a accordé en acceptant d'être examinateur de ce travail malgré ses nombreuses occupations. Je le remercie pour toute l'attention qu'il a portée à ce travail.

Mes remerciements sincères vont à Monsieur Khalid HADDOUCH, Professeur à l'ENSA de Al-Hoceima, pour l'intérêt qu'il a porté à l'égard de cette recherche en s'engageant à être examinateur. Je le remercie aussi pour le temps qu'il a consacré et pour son déplacement de loin.

Je tiens à remercier également Monsieur le Doyen, Monsieur le Vice-Doyen et tout le personnel de la Faculté des Sciences et Techniques de Fès qui ont aidé à la réalisation de cette thèse.

Un remerciement spécial à tous mes collègues docteurs et doctorants du Laboratoire : Modélisation et Calcul Scientifique, pour leurs sympathies, gentillesses, soutiens et discussions enrichissantes tout au long des années que nous avons passées ensemble.

Je n'aurais jamais pu réaliser ce travail doctoral sans le grand soutien moral et matériel de ma famille. Je leur adresse mes grands remerciements de m'avoir offert la liberté dans le choix de mes études, pour la confiance qu'ils m'ont accordée, et pour les sacrifices qu'ils ont consentis pour ma formation et mon bien-être. Le mot merci ne peut pas exprimer à quel point je vous suis reconnaissant. Mes remerciements chaleureux s'adressent à mes sœurs et frères pour leur soutien de prêt et de loin. J'espère qu'ils trouvent dans ce travail l'expression de mon profond amour, mes vœux de succès, de réussite et de bonheur dans leur vie. Avec une grande joie je remercie toute personne que j'ai rencontrée dans ma vie et qui a contribué, directement ou indirectement, à la réalisation de cette thèse. Très heureux d'avoir partagé avec vous des moments soit agréables ou désagréables qui m'ont beaucoup appris.

Résumé

Jusqu'à présent, l'intelligence artificielle a toujours été le domaine le plus débattu, rêvé et poursuivi. Dans ce cadre, de nombreux travaux de recherche ont soulevé la problématique quant à la possibilité de réaliser un programme informatique capable d'imiter le comportement humain et, dans ses réalisations les plus avancées, d'apprendre ou de prendre des décisions. Actuellement, c'est un domaine riche des techniques permettant de concevoir des systèmes intelligents capables de résoudre des problèmes NP-difficiles, notamment des problèmes liés à l'optimisation dans divers domaines : base de données, traitement d'image, théorie des graphes. Dans la première contribution, nous proposons un système portant sur la modélisation du problème étudié en termes de satisfaction de contraintes pondérées et le plan multiple d'exécution des vues. Pour résoudre le modèle obtenu, nous utilisons les algorithmes génétiques tout en adaptant des opérateurs originaux. En effet, la solution consiste à représenter la requête sous forme d'un graphe et rassembler toutes les requêtes pour construire l'espace de recherche optimal. Grace à la modélisation par la satisfaction de contraintes pondérées, nous avons utilisé les algorithmes génétiques pour sélectionner les vues qui réalisent un compromis entre les coûts de traitement de requêtes, maintenance et de stockage dans le but d'accélérer l'exécution des requêtes analytiques dans un entrepôt de données. Dans la deuxième contribution, nous proposons un réseau de neurones récursifs d'architecture originale pour résoudre le problème de stable maximum dans les graphes complexes tel que le plan multiple d'exécution des vues. Pour résoudre l'équation différentielle régissant l'évolution du réseau ainsi conçu, nous utilisons plusieurs méthodes de discrétisation et ce pour justifier le choix de la méthode d'Adams-bashforth à plusieurs pas qui représente des avantages par rapport à la méthode classique. Dans la troisième contribution, nous avons proposé une nouvelle approche pour résoudre les problèmes de satisfaction maximale de contraintes. Cette approche se compose de deux étapes intéressantes : La proposition d'un nouveau modèle du problème de satisfaction maximale de contraintes comme un problème de programmation quadratique sous des contraintes linéaires et l'utilisation de l'algorithme génétique pour résoudre ce dernier modèle.

La diversité des approches proposées pour la sélection des vues et l'optimisation les critères de performance, représentent des points forts qui caractérisent nos travaux de recherche. Les résultats obtenus après la validation des approches proposées dans cette étude sont encourageants et prometteurs pour les problèmes d'optimisations.

Abstract

Artificial intelligence is a field of research it is the most debated, dreamed and pursued domain. In this context, numerous studies have raised the problem of the possibility of carrying out a computer program capable of imitating human behavior and, in its most advanced achievements, of learning or making decisions. Currently, artificial intelligence is a rich field of techniques for designing intelligent systems capable of solving NP-hard problems, including problems related to optimization in various fields: database, image processing and theory of graphs. In our first contribution, we propose a system based on modelling of the studied problem in terms of weighted constraints satisfaction problems for the multiple execution plan of views. To solve the obtained model, we use genetic algorithms with adapting original operators. Indeed, the solution is to represent the query in the form of a graph and gather all the queries to build the optimal search space. Thanks to the modelling by the satisfaction of weighted constraints, we used the genetic algorithm to select the views that realize a compromise between the costs of processing requests, maintenance and storage in order to accelerate the execution of queries. In the second contribution, we propose a recursive neural network with an original architecture to solve the problem of maximum stable in complex graphs such as the multiple execution plane of views. To solve the differential equation governing the evolution of the network thus conceived, we use several discretization methods to justify the choice of the Adams-bashforth method with several steps that represents advantages in comparison with a classical method. In the third contribution, we propose a new approach to solve the problems of maximum satisfaction of constraints. This approach consists of two interesting steps: The proposition of a new model of the maximal constraints satisfaction problems like a problem of quadratic programming under linear constraints and use the genetic algorithm to solve this model.

The diversity of approaches proposed for the selection of views and optimization represent strengths that characterize our research work. The results obtained after the validation of the approaches proposed in this study are encouraging and promising for optimization problems.

Travaux réalisés

Cette thèse est un fruit de trois publications dans des revues internationales et quatre communications internationales.

Articles

- Mohammed EL ALAOUI, Karim EL MOUTAOUAKIL and Mohamed ETTAOUIL

'*A hybrid approach to solving the view selection problem in data warehouse*', International Journal of Computer Sciences and Engineering', Volume 6, pp.270-275, 2018.

- Mohammed EL ALAOUI, Karim EL MOUTAOUAKIL and Mohamed ETTAOUIL

'*A Multi-Step Method to Calculate the Equilibrium Point of the Continuous Hopfield Networks: Application to the Max-Stable Problem*', WSEAS Transactions on Systems and Control, Volume 12, pp. 418-425, 2017

- Mohammed EL ALAOUI, Karim EL MOUTAOUAKIL and Mohamed ETTAOUIL

'*Weighted Constraint Satisfaction And Genetic Algorithm To Solve The View Selection Problem*', International Journal of Database Management Systems, Volume 9, pp. 11-20, 2017.

Communications

- Fidae HARCHLI, Abdelatif ESSAFI, Mohammed EL ALAOUI and Mohamed ETTAOUIL, '*Proposition and Resolution of New Mathematical Model to Optimize the MLP Architecture Using GA optimization*', Conférence Internationale Francophone sur la Science des Données, Marrakech.
- Mohammed EL ALAOUI, Mohamed ETTAOUIL, '*Problème de satisfaction maximal de contraintes et les algorithmes génétiques*', quatrième rencontre du calcul scientifique et application aux problèmes socio-économique, FST, Fès.
- Mohammed EL ALAOUI, Mohamed ETTAOUIL, '*La modélisation des problèmes sous forme de problème de satisfaction de contraintes*', troisième rencontre du laboratoire modélisation et calcul scientifique, FST, Fès.
- Khalid HADDOUCH, Mohammed EL ALAOUI, Mohamed ETTAOUIL, '*Genetic algorithms for solving the maximal constraint satisfaction problems*' International conference on intelligent information and network technology, Settat.

Table de matières

Nomenclatures

AG	Algorithme génétiques
AE	Algorithmes d'escalade
CSP	Problème de satisfaction de contraintes
CSOP	Problème de satisfaction de contraintes et d'optimisation sous contraintes
WCSP	Problème de satisfaction de contraintes pondérées
Max-CSP	Problème de satisfaction maximale des contraintes
DAG	Graphe orienté acyclique
ED	Entrepôt de données
RN	Réseau de neurones
PQ	Problème quadratique
SCSP	Problème de satisfaction de contraintes symboliques
SGBD	Système de gestion de base de données
SQL	Langage de requête structurée
TPC-H	Transaction Processing Performance Council

Liste de figures

Liste de tableaux

Introduction générale

Depuis l'aube de l'informatique, l'intelligence artificielle a toujours été le domine le plus étudié, rêvé et poursuivi. De nombreux travaux ont soulevé la problématique quant à la possibilité de réaliser un programme informatique capable d'imiter le comportement humain et, dans ses réalisations les plus avancées, d'apprendre ou de prendre des décisions. Actuellement, l'intelligence artificielle est un domaine riche des techniques permettant de concevoir des systèmes intelligents capables de résoudre des problèmes NP-difficiles, notamment des problèmes liés à l'optimisation dans divers domaines : base de données, traitement d'image, théorie des graphes etc.[1].

Cette thèse entre dans le cadre de l'optimisation de performance de requêtes dans un entrepôt de données, la satisfaction des contraintes et les réseaux de neurones artificiels. Dans un environnement de plus en plus dynamique et complexe, il est nécessaire que les entreprises recueillent des informations du monde extérieur pour comprendre les tendances d'affaires qui ont un impact sur l'entreprise. Ces informations permettent aux entreprises de se positionner en décidant d'une politique appropriée sur le marché. Par conséquent, la nécessité d'une information rapide et complète encourage la recherche sur les mécanismes permettant de stocker et manipuler les données de manière appropriée, afin de rationaliser le processus de prise de décision. Les travaux de recherche menus dans ce contexte ont donné naissance à des systèmes d'aide à la décision. Ces systèmes sont des outils interactifs conçus pour aider les décideurs à extraire des informations utiles à partir de big data pour prendre des décisions. Dans ce sens, l'entrepôt de données se distingue comme technologie de système d'aide à la décision. Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision[2]. Il intègre de grandes quantités de données provenant de diverses sources. Les données de l'entrepôt sont organisées par des schémas qui facilitent le stockage et le traitement de l'information. Cette représentation en entrepôt permet aux utilisateurs d'exécuter plus efficacement des rapports, des analyses et des requêtes sur les données stockées.

Les requêtes sont traitées à partir des opérations de sélection, de jointure et d'agrégation effectuées sur la base de données. Ces deux dernières opérations représentent d'importants consommateurs de temps et de ressources de traitement[3]. Surtout lorsqu'elles sont appliquées à de gros volumes de données. Cependant, les requêtes doivent retourner une

réponse dans un court laps de temps. Par conséquent, plusieurs solutions ont été proposées dans la littérature pour maximiser les performances du traitement des requêtes[4]. Des études récentes utilisent des solutions telles que la fragmentation verticale, horizontale ou hybride pour augmenter la vitesse d'exécution des requêtes [5]. Dans le même objectif, la matérialisation est une autre alternative permettant d'offrir des réponses rapides aux requêtes [3]. Les vues matérialisées stockent des données pré-calculées pour résoudre le problème de la surcharge associée aux opérations de jointure coûteuses et aux agrégations de données requises par les requêtes. Ainsi, compte tenu du coût de stockage, il est nécessaire de sélectionner les meilleures vues à matérialiser, c'est-à-dire compatibles avec les exigences de stockage et offrant moins de temps de réponse pour traiter les requêtes. Cependant, dans la littérature, la plupart des études traitant le problème de sélection de vues (matérialisation) ne couvrent pas le coût de stockage [6]. C'est la raison pour laquelle nous avons fait le choix d'étudier ce problème d'optimisation de performance des requêtes en base de données massives via les techniques de machine Learning et programmation par contraintes.

Dans la présente thèse, nous utilisons des techniques performantes de l'intelligence artificielle pour résoudre le problème de la sélection des vues dans un entrepôt de données. Ces techniques sont la modélisation par satisfaction des contraintes, la théorie des graphes, les réseaux de neurones artificiels et les algorithmes évolutionnistes.

Dans un premier temps, nous proposons un système portant sur la modélisation du problème étudié en termes de satisfaction de contraintes pondérées et le plan multiple d'exécution des vues[7]. Pour résoudre le modèle obtenu, nous utilisons les algorithmes génétiques tout en adaptant des opérateurs originaux. En effet, la solution consiste à représenter la requête sous forme d'un graphe et rassembler toutes les requêtes pour construire l'espace de recherche optimal. Grace à la modélisation par la satisfaction de contraintes pondérées, on a fait appel à l'algorithme génétique pour sélectionner les vues qui réalisent un compromis entre les coûts de traitement de requêtes, de maintenance et de stockage dans le but d'accélérer l'exécution des requêtes.

Dans un deuxième temps, nous proposons un réseau de neurones récursifs d'architecture originale pour résoudre le problème de stable maximum dans les graphes complexes tel que le plan multiple d'exécution des vues[8]. Pour résoudre l'équation différentielle régissant l'évolution du réseau ainsi conçu, nous utilisons plusieurs méthodes de discrétisation et ce pour justifier le choix de la méthode à plusieurs pas qui représente des avantages par rapport à la méthode classique.

Nous avons testé les approches proposées sur des bases de données académiques. Dans ce sens, ces approches ont montré une performance remarquable dans la résolution du problème de sélection des vues connu par son caractère NP-complet[9]. En effet, les vues sélectionnées ont permet un gain en termes du temps, de mémoire et de la qualité des solutions.

Dans le troisième temps, nous proposons une nouvelle approche pour résoudre les problèmes binaires de satisfaction maximale de contraintes (Max-CSP). Dans cette nouvelle approche, le problème à résoudre est décrit en termes d'un problème de programmation quadratique sous des contraintes linéaires. Ce problème a été résolu via l'algorithme génétique. Les résultats obtenus après la validation de notre approche, proposée dans cette étude, sont encourageants et prometteurs pour les problèmes Max-CSP.

La présente thèse s'articule autour de quatre chapitres suivants : le premier chapitre présentera un aperçu sur l'architecture de l'entrepôt de données et sa modélisation multidimensionnelle. Nous représentons différentes principales techniques d'optimisation de performance des requêtes telle que la fragmentation et la matérialisation de vues. Nous verrons les différents espaces de recherches, comme par exemple ceux à base de : cube en treillis multidimensionnel, et à base de graphe de vues (ET-OU), de plan multiple d'exécution des vues et de fouille de données ainsi qu'une analyse sur les travaux réalisés dans ce contexte.

Nous présenterons dans un second plan les réseaux de neurones récurrents discrets et continus en détaillant leur contexte historique ainsi que ses caractéristiques générales. Nous verrons également de manière très globale leur fonctionnement en nous plaçons sur différent moment de sortie du système. En effet, la sortie du réseau de neurones dépend non seulement des entrées courantes et des poids de connexion, mais aussi des sorties offertes par le réseau dans les moments précédents. La deuxième partie de ce chapitre, présentera les algorithmes génétiques comme une autre technique d'optimisation issue du paradigme de l'intelligence artificielle. Les différentes étapes de cet algorithme comme représentation génétique, population initiale, fonction d'adaptation, mécanisme de sélection et la convergence seront élaborées.

Nous introduirons dans le troisième chapitre le problème de satisfaction de contraintes que nous illustrerons à travers quelques exemples classiques. Nous allons également décrire les principales méthodes de résolution utilisées actuellement. Puis, nous étendrons la

problématique de satisfaction de contraintes aux problèmes : de satisfaction maximale de contraintes, d'optimisation de contraintes et de satisfaction de contraintes pondérées.

Nous nous focaliserons dans le dernier chapitre sur les différentes contributions que nous avons proposées pour résoudre la problématique d'optimisation des requêtes sur des bases de données massives, la problématique du stable maximum et les problèmes de satisfaction maximal de contraintes.

Pour décrire la première contribution, nous commencerons par la description de la benchmark TPC-H, suivis par les différentes démarches permettant de construire l'espace de recherche optimal. Nous présenterons ensuite la modélisation sous forme de problème de satisfaction de contraintes pondérées ainsi que la résolution par l'algorithme génétique pour choisir un ensemble optimal des vues à matérialiser. Enfin, nous développerons un ensemble d'expériences pour valider la performance de notre approche.

Dans la deuxième contribution, nous viserons aussi à résoudre le problème de stable maximum par une nouvelle approche du réseau de Hopfield. Pour se faire, nous présenterons en premier lieu la formulation et la modélisation du problème sous forme d'un programme mathématique quadratique. En deuxième lieu, nous définirons l'architecture du réseau de Hopfield. Par la suite, nous introduirons la nouvelle approche de discrétisation de l'équation différentielle caractérisant le réseau de Hopfield. Pour évaluer notre approche, nous discuterons nos résultats en terme de convergence comparées avec ceux du réseau de Hopfield classique. Dans la troisième contribution, nous proposerons une nouvelle approche pour résoudre les problèmes binaires de satisfaction maximale de contraintes (Max-CSP). Dans cette nouvelle approche, le problème à résoudre est décrit en termes d'un problème de programmation quadratique sous des contraintes linéaires. Ce problème a été résolu via l'algorithme génétique. Les résultats obtenus après la validation de notre approche, proposée dans cette étude, sont encourageants et prometteurs pour les problèmes Max-CSP. Nous clôturons le chapitre par une conclusion générale et perspectives.

Chapitre 1 : Optimisation de performance des requêtes dans un entrepôt de données

1.1 Introduction

Depuis la naissance de l'univers informatique, les organisations ont créé des systèmes pour stocker les données générées dans les transactions commerciales. Initialement, les données n'étaient pas considérées comme une ressource stratégique, elles n'ont donc été stockées que pour tenir des registres qui rendent compte de l'évolution quotidienne de l'entreprise[10]. Au fil de temps, des améliorations continues dans le matériel ont permis de stocker de grandes quantités de données et d'augmenter la puissance de traitement de celles-ci. Grâce à cela, de nombreuses institutions ont commencé à conserver leurs données transactionnelles dans le but d'être utilisées pour des prédictions commerciales et des analyses rétrospectives[11]. Dans ce contexte, il existe des systèmes OLTP (Online Trasaction Processing), qui entrent dans la catégorie des systèmes d'information transactionnels. Ce type a pour fonction principale de gérer les activités quotidiennes des entreprises, tels que la réservation de chambres d'hôtel, le traitement de la paie, le contrôle des stocks, les ventes de supermarchés, les transactions bancaires, etc. Les autres caractéristiques des systèmes transactionnels sont les suivantes: avoir un degré de détail élevé de l'information, être volatile et se concentrer sur les transactions qui se produisent à un moment donné. Cependant, ces systèmes sont limités au niveau de la prise de décision car les requêtes historiques ont un impact négatif sur le fonctionnement de ces systèmes.

Parallèlement aux systèmes transactionnels, les systèmes décisionnels sont axés sur le traitement de gros volumes de données et leur analyse, en fournissant des réponses plus rapides et plus flexibles aux requêtes des utilisateurs que les systèmes transactionnels. Ces

systèmes décisionnels permettent de collecter et d'organiser les informations analytiques nécessaires, en les plaçant sous différents formats, tels que des tableaux, des graphiques, des rapports, etc. Les principales caractéristiques de cette technologie sont :

- Vitesse : Etant donné un grand volume de données, la vitesse de traitement des requêtes afin de fournir des informations souhaitées est un point crucial à prendre en considération. En revanche aux systèmes OLTP, les systèmes décisionnels ont montré une grande rapidité dans des situations pareilles.
- Analyse : Les systèmes décisionnels sont dédiées à faire des analyses sur les données historisées dans des entrepôts de données afin de mettre entre les mains des décideurs les connaissances nécessaires pour prendre des décisions ou appliquer des plans stratégiques. Ces analyses s'étalent sur les données pour faire sortir les informations et par la suite sur celles-ci pour obtenir lesdites connaissances.
- Multidimensionnel : Ce concept de multidimensionnel est expliqué par la nécessité d'obtenir ou de visualiser des résultats lors de traitements de requêtes suivant plusieurs critères simultanément. Ainsi, si un utilisateur de système décisionnel veut savoir les ventes réalisés sur des produits par rapport au temps, il sera censé de choisir trois critères qui sont PRODUIT, VENTE et TEMPS. Cette requête qui parait simple, exige un temps important et un code compliqué dans les systèmes OLTP, or elle est assez simple et facile à traiter dans les systèmes décisionnels vus le mécanisme multidimensionnel qu'ils disposent.

En définissant l'entrepôt de données comme une base de données spéciale, il est important de faire une comparaison entre les bases de données transactionnelle et l'entrepôt de données afin de comprendre pourquoi elle est considérée comme spéciale. Les principales différences entre ces deux environnements sont présentées dans le tableau 1.1 [12] :

Variables	**BD transactionnelle**	**Entrepôt de données**
Utilisateur	Opérationnel	Décideur
Accès	Élevé	Moyen et bas
Temps de réponse	Secondes	Secondes à minutes
Stabilité	Dynamique	Statique jusqu'à son actualisation
Structure BD	Entité-association	Multidimensionnel
Taille BD	100 Mo-Go	100 Go-TB

Tableau 1.1 Comparaison entre la base de données transactionnelle et l'entrepôt de données

Le scénario actuel dans lequel les entreprises évoluent nécessite des systèmes capables de fournir des informations analytiques fiables et rapides aux décideurs. Or, la plupart des systèmes OLTP actuels ne répondent qu'à des zones très spécifiques, prennent beaucoup de temps pour répondre aux besoins complexes de l'utilisateur et ne garantissent pas la confiance dans les informations analytiques. La solution est apportée avec l'arrivée des systèmes décisionnels OLAP pour Online Analytical Processing en anglais, qui utilisent des entrepôts de données (DataWarehouse en anglais). L'architecture de ceux-ci est né avec l'idée de fournir des informations propres, consolidées et fiables, avec un accès rapide et orienté vers l'utilisateur final et qui est dotée d'un référentiel de données qui permet d'obtenir des informations complètes, correctes, cohérentes, opportunes et accessibles, en fonction des besoins des utilisateurs finaux dans les délais et formats appropriés. Compte tenu de ces caractéristiques, l'entrepôt de données se traduit comme un système qui prend en charge le processus de prise de décision, ce qui génère de la confiance parce que l'information est fiable[13].

Ce chapitre présentera un aperçu sur l'architecture de l'entrepôt de données et sa modélisation multidimensionnelle. Nous représenterons différentes principales techniques d'optimisation de performance des requêtes telle que la fragmentation et la matérialisation de vues. Nous verrons les différents espaces de recherches, comme par exemple ceux à base de: cube en treillis multidimensionnel, et à base de graphe de vues (ET-OU), de plan multiple d'exécution des vues et de fouille de données. Nous finirons ce chapitre par une analyse sur les différents travaux réalisés pour l'optimisation des requêtes.

1.2 Entrepôt de données

Un entrepôt de données (ED) est une base de données qui stocke des informations pour la prise de décision. Cette information est construite à partir de bases de données qui enregistrent les transactions des entreprises de bases de données opérationnelles[14]. L'objectif des ED est de consolider les informations provenant des différentes bases de données opérationnelles et de les rendre disponibles pour la réalisation d'analyses de données. Les données de l'ED sont le résultat de transformations, de contrôles de qualité et d'intégration de données opérationnelles. La priorité dans les ED est l'accès interactif et immédiat à l'information stratégique d'un secteur d'activité. Les utilisateurs peuvent effectuer leurs propres requêtes sur des données, en utilisant des outils spécialisés avec des interfaces graphiques.

Les caractéristiques des ED font que les stratégies de conception des bases de données opérationnelles ne sont généralement pas applicables à la conception des ED. Les modèles de données pour spécifier les données stockées dans l'ED sont différents. Au niveau conceptuel, des modèles multidimensionnels représentent les informations sous forme de tables de faits et les tables de dimensions. De l'architecture aux requêtes exécutées nous passons par une nouvelle conception d'entrepôt de données[15].

1.2.1 Architecture d'entrepôt de données

Etre capable de transformer les données en connaissances est un processus complexe. En effet, un entrepôt de données est beaucoup plus qu'une simple copie de données d'un endroit à un autre, c'est-à-dire des systèmes opérationnels à une base de données indépendante. Un entrepôt de données est avant tout une architecture qui doit servir d'infrastructure pour apporter une solution complète au problème précité. L'architecture d'un entrepôt de données est représentée à la figure (1.1).

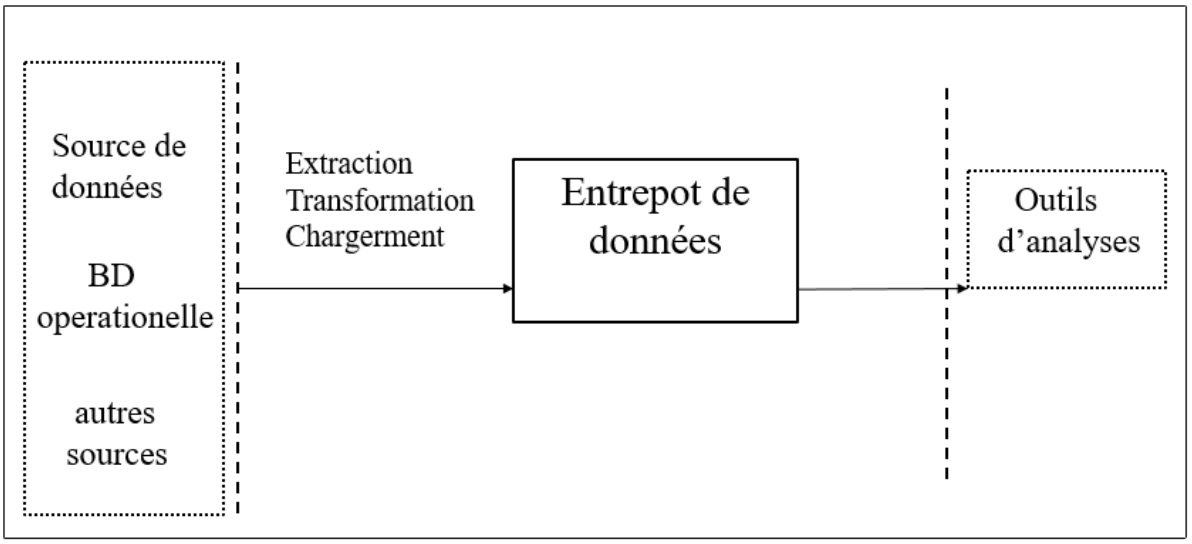

Figure 1.1 Composants de l'architecture de l'entrepôt de données

a) Sources de données

Ce composant est celui qui est normalement présent à l'origine dans les entreprises à partir duquel la capture de données qui sera envisagée dans l'entrepôt de données est effectuée. Ces sources de données peuvent être des systèmes opérationnels d'entreprise (représentent l'environnement à partir duquel la plupart des données importantes de l'exploitation

quotidienne de la société sont obtenues) et des sources externes telles que copies de sauvegarde, tableurs, fichiers plats de l'entreprise, résultats d'études de marché, etc[16].

b) Extraction, transformation et chargement

Pour remplir l'entrepôt de données, les blocs de données doivent être déplacés, souvent à partir de différents systèmes d'exploitation, structures de fichiers et bases de données, via des processus planifiés fréquemment exécutés en dehors des heures de travail afin de ne pas perdre de temps. Les sous-systèmes destinés à alimenter l'entrepôt de données peuvent être construits à l'aide d'outils et de produits disponibles sur le marché, de programmes et de processus codés de toutes pièces ou de combinaisons de ces éléments. Ceux-ci permettent d'assurer la croissance évolutive de l'entrepôt de données, offrant une évolutivité et un support pour de grandes quantités de données et des requêtes complexes. Des difficultés supplémentaires peuvent être rencontrées en fonction des sources de données disponibles, ce qui implique l'utilisation de différents outils technologique pour accéder à chacun d'entre eux[17].

- **Extraction**

L'objectif principal de la phase d'extraction est de capturer et de copier les données requises d'un ou de plusieurs systèmes opérationnels ou sources de données. Les données extraites sont placées dans un fichier intermédiaire avec un format défini, qui sera ensuite utilisé par la phase suivante du processus. Les enregistrements rejetés dans le processus doivent être enregistrés dans un fichier ou un journal des rejets afin qu'ils puissent être analysés plus tard et avoir ainsi la possibilité de les charger correctement dans l'entrepôt de données. En outre, cela permet de découvrir les erreurs qui se sont produites dans les processus de création de données opérationnelles. Des exemples de ces erreurs sont les violations d'intégrité, les clés en double, les formats de données incorrects et les données non valides telles que les champs vides, les dates futures et les montants négatifs etc[18].

- **Transformation**

Les fonctions de base à exécuter dans cette phase consiste à lire les fichiers intermédiaires générés par la phase d'extraction, effectuer les transformations nécessaires, construire les enregistrements dans le format d'entrepôt de données et créer un fichier de sortie contenant tous les nouveaux enregistrements à charger dans l'entrepôt de données.

- **Chargement**

L'objectif de cette phase est de prendre les enregistrements formatés par la phase de transformation et de les charger dans l'entrepôt de données, qui est le conteneur de toutes les données d'information actuelles et historiques requises par les opérations sur l'entrepôt de données.

c) Outils d'analyse

Sans les bons outils d'accès et d'analyse, l'entrepôt de données peut être vu comme un entrepôt de données sans aucune utilisation. Dans ce fait, il est nécessaire d'avoir des techniques qui capturent rapidement des données importantes et qui peuvent être analysées de différents points de vue, mais qui doivent également transformer les données capturées en informations utiles pour l'entreprise. Ce bloc comprend également le matériel et le logiciel impliqués dans la visualisation et la publication des rapports d'impression, des tableurs, des graphiques et des diagrammes pour l'analyse et la présentation[15].

1.2.2 Modélisation multidimensionnelle

Pour comprendre l'un des aspects les plus importants de l'architecture de l'entrepôt de données, comme la modélisation des données, il est nécessaire de présenter les différences entre les deux mondes de modélisation existants : entité-association et multidimensionnelle. Généralement pour créer un seul modèle complexe de tous les processus d'une organisation, le modèle entité-association s'est avéré efficace pour créer des systèmes de traitement de transaction en ligne (OLTP). D'autre part, le modèle multidimensionnel crée son modèle pour refléter les processus métier discrets. Ce modèle organise les informations en structures correspondant à la manière dont les analystes interrogent les données d'entrepôt de données. L'objectif de la conception multidimensionnel est de fournir à l'utilisateur final un moyen simple de représenter l'information analytique. Il existe principalement trois schémas pour le modèle multidimensionnel : le modèle en étoile, le modèle en flocon de neige et le modèle en constellation[19].

a) Modèle en étoile

Le modèle en étoilé est nommé ainsi en raison de son apparence physique qui ressemble à une étoile. Ce modèle est composé d'une table centrale qui contient une clé primaire composite, cette table est appelée table des faits et d'un ensemble de tables périphériques

nommées tables de dimensions. Chacune des tables de dimension possède une clé primaire qui correspond exactement à l'un des composants de la clé composite de la table de faits. Les tables de faits, en plus de leurs champs clés, contiennent une ou plusieurs mesures ou indicateurs. La figure 1.2 présente un exemple de modèle en étoile[20].

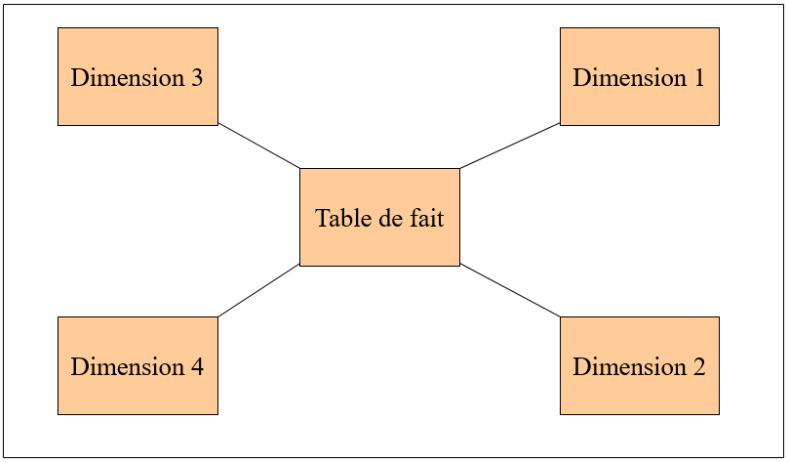

Figure 1.2 Exemple de modele en étoile

b) Modèle en flocon de neige

Le modèle en flocon de neige s'est inspiré du modèle en étoile. Il se distingue par le fait que les dimensions peuvent avoir des connexions avec d'autres dimensions, c'est-à-dire qu'il existe une relation entre les tables dimensions. Cette extension du modèle en étoile dans laquelle les tables dimensionnelles sont normalisées. Les avantages de cette modélisation sont la réduction de la taille et de la redondance dans les tables de dimension et l'augmentation de la flexibilité dans la définition des dimensions, mais l'augmentation du nombre de tables nécessite plus d'opérations de jointure pour répondre aux requêtes, ce qui aggrave le temps d'exécution des requêtes. Un autre inconvénient est la maintenance requise par les tables dimensionnelles[15].

c) Modèle en constellation

Ce modèle est plus complexe que les autres modèles (en étoile et flocon de neige). Il consiste en un mélange de ces deux derniers. Ainsi, il contient plusieurs tables de faits et tables de dimensions. Avec cette solution, les tables de dimensions peuvent être partagées entre plusieurs tables de faits. Le modèle en constellation à beaucoup de flexibilité, cependant, son

problème majeur est la complexité et difficulté de maintenance vu le grand nombre de tables existantes.

1.2.3 Structure de base de requête analytique

La structure de base d'une expression en SQL pour une requête analytique est composée de quatre clauses [21]:

SELECT <Regroupement d'attributs et/ou de fonctions d'agrégation>
FROM <Table de faits>$D_1, D_2, \dots, D_k$
WHERE <Contraintes>
GROUP BY <Attributs de regroupement>

Les requêtes analytiques incluent généralement des restrictions sur plusieurs tables de dimensions qui sont à leurs tours en relation avec la table de fait via des clés étrangères. Les termes $D_1, D_2, \dots, D_k$ sont les tables de dimensions. Dans la clause SELECT, le calcul effectué est basé sur l'attribut de mesure numérique dans la table de faits et dans la clause GROUP BY, chaque attribut est un niveau d'agrégation choisi de dimensions.

1.3 Techniques d'optimisation de performance des requêtes

Un entrepôt de données contient une quantité importante de données. Son succès dépend donc aussi de l'efficacité avec laquelle elles sont gérées. Il existe des techniques d'optimisation des performances qui permettent à l'entrepôt de données de répondre aux besoins des utilisateurs[22]. Un entrepôt de données peut être optimisé avec certaines techniques comme :

- La fragmentation est une technique de conception permettant de diviser une base de données en un ensemble de partitions ou de fragments de manière à ce que la combinaison des partitions produise la base de données originale sans perte ni ajout d'information [5]. Le résultat du processus de la fragmentation est un ensemble de fragments définis par un schéma de fragmentation. Elle peut être verticale, horizontale ou bien hybride : La fragmentation verticale divise une table en un sous-ensembles d'attributs, chaque sous-ensemble forme un fragment vertical. La fragmentation horizontale divise une table en un sous-ensembles de tuples, chaque sous-ensemble

formant un fragment horizontal. Alors que la fragmentation hybride consiste à appliquer une fragmentation verticale et horizontale simultanément dans une table.

- La matérialisation est une structure qui permet un accès rapide aux données par la présence de certaines vues matérialisées et par conséquence peut significativement améliorer la performance de requêtes, notamment pour les applications de grande taille. Et elles sont utiles lorsqu'une requête peut récupérer des données au lieu d'avoir calculer le résultat de la requête au moment de l'exécution[5].

Pour que la vue matérialisée soit utilisée, elle doit suivre les données de la table de base. Ainsi, il est nécessaire de recréer la vue à partir des données sources ou de la mettre à jour dès que la table originale change, là on parle de coût de maintenance. D'autres coûts importants qui concernent la matérialisation comme les coûts de traitement de requêtes et de stockage sont impliqués dans le processus de matérialisation. La technique de matérialisation sera développée pour but d'optimiser la performance de requêtes.

1.3.1 Matérialisation des vues

Avec le terme entrepôt de données (ED), nous entendons une archive capable de mémoriser des volumes d'informations[15], qui sera ensuite utilisée au niveau d'analyse et comme support pour l'élaboration de choix stratégiques par les entreprises[23]. Compte tenu de l'énorme quantité de données normalement présente dans un ED, l'un des problèmes les plus urgents dans leur utilisation est le coût d'exécution des requêtes utilisées pour l'extraction de données[24]. Pour éviter ce problème, il est possible d'introduire l'utilisation de vues matérialisées. L'idée de base est de rendre directement disponible à l'application qui utilise le système ED un prétraitement des données dont elle a besoin. Les vues matérialisées permettent de stocker le résultat du traitement d'une requête sur un support physique dans le but de pouvoir l'utiliser plus tard. Nous devons alors être conscients que ni l'espace où les vues peuvent se matérialiser, ni le temps de leur maintenance sont illimités, nous sommes confrontés à un autre problème celui du choix sur lequel il vaut mieux matérialiser. En ce qui concerne les vues matérialisées, trois cas possibles peuvent se présenter :

- Matérialisation complète

Elle est considérée comme un choix idéal. Dans ce cas, nous comptons enregistrer toutes les vues possibles sur le disque dur. En terme du temps de réponse aux requêtes, nous obtenons

une accélération maximale car, pour chaque requête entrante, il existe un résultat pré-calculé qui peut être utilisé pour répondre à la requête. Cependant, dans la pratique, cette option n'est pas possible dans certains systèmes, car le stockage de toutes les vues possibles peut nécessiter beaucoup d'espace disque, ce qui ne peut pas être pris en charge par ces systèmes. De plus, la matérialisation de toutes les vues entraînera un long processus de mise à jour.

- Matérialisation partielle

Dans la matérialisation partielle, un seul sous-ensemble de toutes les vues possibles est sélectionné pour être matérialisé. Par conséquent, un équilibre peut être réalisable entre le temps de réponse de la requête, le temps de mise à jour et la taille de l'espace disque.

- Pas de matérialisation

En utilisant cette option, aucune vue n'est matérialisée. Bien que la quantité d'espace disque utilisée pour stocker les vues soit égale à zéro et qu'aucun processus de mise à jour ne soit nécessaire, cette option présente les performances les plus faibles en terme du temps de réponse.

1.3.2 Coût de traitement de requête

Pour un ensemble des vues matérialisées (M), le temps total nécessaire pour évaluer toutes les requêtes possibles dans le système peut être calculé comme suit [9] :

$$Cost_Q = \sum_{q \in Q} f_q . Cost_q(M)$$

Le $Cost_q$ indique le coût nécessaire au traitement de la requête q en présence de M. Où :

Q est l'ensemble des requêtes,
q est une requête particulière,
f_q est la fréquence de la requête q.
Cette équation s'applique, tant que certaines requêtes sont souvent posées alors que d'autres requêtes se produisent rarement.

1.3.3 Coût de maintenance

Les vues matérialisées fonctionnent comme un système de cache dans le domaine de la gestion de la mémoire. Dans un système de cache, les données fréquemment consultées dans la mémoire principale sont dupliquées dans un stockage rapide et de petite capacité. Toute demande future pour les données qui ont déjà été mises en cache se fait par le cache plutôt que par référence à la mémoire principale. Cela réduit le temps d'accès[25]. Dans la technique du cache, chaque fois que les données d'origine dans la mémoire principale sont modifiées, les données mises en cache doivent également être mises à jour. De même, dans la sélection de vue matérialisée, chaque fois que l'entrepôt de données est chargé et que les nouveaux enregistrements sont insérés dans la table de faits et entraînent des modifications, les vues dérivées de la table de faits doivent également être mises à jour. Normalement, afin de réduire la perturbation de la fenêtre de travail d'un entrepôt de données, le processus de mise à jour est effectué lorsque le système est inactif [26]. Le coût total de mise à jour pour un ensemble M de vues matérialisées peut être calculé comme suit [27]:

$$Cost_M = \sum_{m \in M} f_u . Cost_m$$

Où f_u est la fréquence de mise à jour de la vue, $Cost_m$ est le coût requis pour mettre à jour la vue matérialisée en présence de M.

1.3.4 Coût de stockage

Le coût total de l'espace disque requis pour sauvegarder un sous-ensemble de vues considérées pour la matérialisation (M) peut être calculé comme suit[28]:

$$\sum_{v \in M} S_v \leq S$$

Où S_v correspond à la taille de l'espace disque requis pour stocker la vue.

1.4. Espaces de recherche

Le choix d'une représentation appropriée est considérée comme un élément clé pour la sélection des vues afin d'optimiser le coût d'une requête, en vue de coût d'entretien, ou les deux. Les différents espaces de recherches bien connus ont été proposés tels que les cube en treillis multidimensionnel[9], [27], [29]–[34], graphe de vues ET-OU [3], [35]–[37], plan

multiple d'exécution des vues[38]–[40], et espace de recherche à base de fouille de données[41].

1.4.1 Espace de recherche à base de cube en treillis multidimensionnel

Généralement, les données dans les entrepôts de données sont conceptualisées comme cubes de données multidimensionnels où chaque cellule du cube de données est une vue consistant en une agrégation d'intérêt[26]. Ce cube en treillis a été utilisé pour exprimer les dépendances entre différentes cellules ou vues du cube de données. Graphiquement, cette représentation de cube en treillis multidimensionnelle est constituée des nœuds représentent les vues possibles susceptibles d'être matérialisées, et les arcs représentent les dépendances entre les vues connectées[26], [42]. L'avantage de cette représentation est que les vues les plus bénéfiques peuvent être trouvées directement à partir des relations de base ou du schéma de l'entrepôt de données sans tenir compte des fichiers journaux de la requête et de la fréquence d'accès à la requête. Cependant, l'inconvénient fondamental de cette représentation de cube en treillis est que le nombre de nœuds dans le schéma croît exponentiellement avec la dimension de l'entrepôt de données.

a) La relation de dépendance entre les requêtes

Chaque nœud de cube en treillis représente une vue/requête de clause GROUP BY. Considérons deux vues v_i et v_j. Nous disons $v_i \leq v_j$ si et seulement si v_i peut-être répondu en utilisant seulement les résultats de la vue v_j. Nous disons alors que v_i dépend de v_j. Par exemple, dans la figure (1.3), la vue (part) peut être résolue en utilisant les résultats de la vue (part, customer). Donc (part) $\leq$ (part, customer). De plus, si des vues ne sont pas comparables entre elles, on utilise l'opérateur $\not\subset$ pour traduire leur indépendance. Par exemple: (part) $\not\subset$ (customer) et (customer) $\not\subset$ (part).

Pour un treillis, deux éléments (vues ou requêtes) doivent avoir au moins une limite supérieure et une plus grande limite inférieure selon la commande $\leq$. Cependant, en pratique, nous n'avons besoin que à des hypothèses suivantes :

- $\leq$ est un ordre partiel,
- un élément supérieur, dont dépend chaque vue.

b) Notation de cube en treillis

Nous dénotons un cube en treillis avec un ensemble d'éléments L (requêtes ou vues) et relation de dépendance $\leq$ par $(L, \leq)$. Les ancêtres et les descendants d'un élément d'un cube en treillis, sont définis comme suit:

- Ancêtre $(v_i) = \{v_j \mid v_i \leq v_j\}$
- Descendant $(v_i) = \{v_j \mid v_j \leq v_i\}$

c) Graphe de cube en treillis

Il est courant de représenter un cube en treillis par un graphe en treillis, un graphe dans lequel les éléments en treillis sont les nœuds. Ainsi, pour deux vues de cube en treillis v_i et v_j, le graphe de cube en treillis a un chemin descendant de v_j vers v_i si et seulement si $v_i \leq v_j$. Par exemple, dans le cube en treillis multidimensionnel de la Figure (1.4), la vue numéro 2 peut être calculée à partir de la vue numéro 1. De même, la vue la plus basse et la plus petite représente une vue qui ne comprend qu'un seul enregistrement. Cet enregistrement est l'agrégation de tous les enregistrements existants dans la table de faits et est la vue la plus résumée. Cette vue peut être calculée à partir de toutes les autres vues du treillis. En fait, le plus petit ancêtre matérialisé d'une vue est utilisé pour répondre à une requête sauf si la vue correspondante à cette requête est matérialisée.

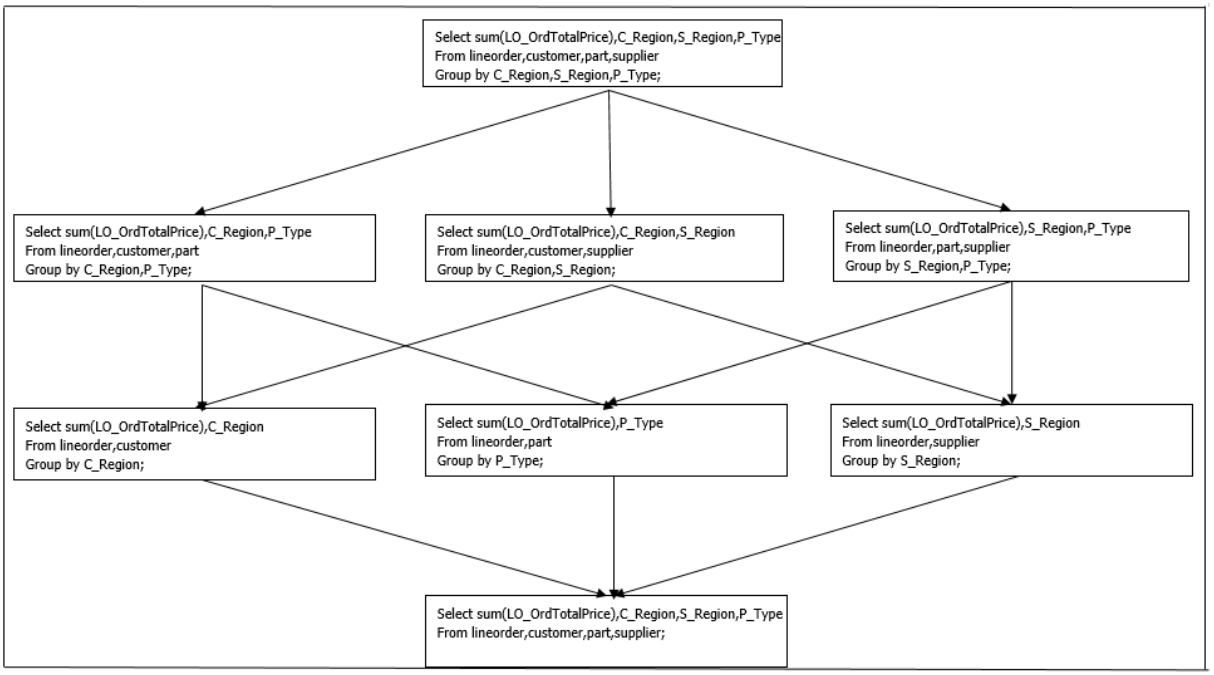

Figure 1.3 Réseau de dépendances organisées à partir de tout groupe possible par requêtes

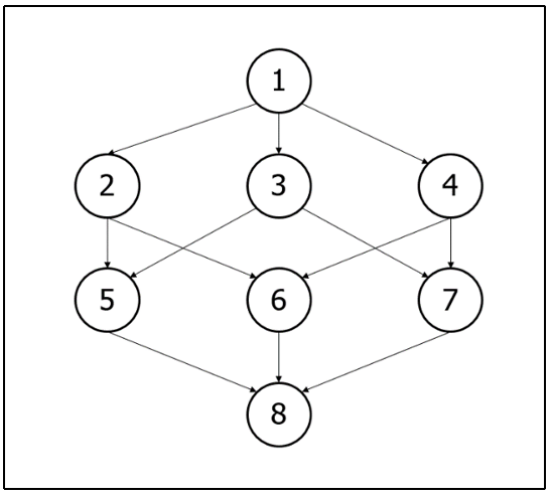

Figure 1.4 Réseau de dépendance correspond à la figure 1.3

1.4.2 Espace de recherche à base de graphe des vues (ET-OU)

L'espace de recherche ET-OU est suggéré comme l'une des représentations du problème de sélection de vue. Les requêtes et les vues sont exprimées en terme de graphes acycliques dirigés (DAG Directed Acyclic Graph en anglais).

L'espace de recherche ET-OU est défini en utilisant les concepts d'arcs ET et OU-DAG. Un ET-DAG pour une requête ou une vue est un graphe acyclique orienté ayant les relations de base comme 'puits' sans arcs sortants et la vue (nœud) V comme 'source' sans arc entrant. Si un nœud ou une vue u a des arcs sortants vers les nœuds $v_1, v_2, \dots, v_k$, toutes les vues $v_1, v_2, \dots, v_k$ sont nécessaires pour calculer le coût de u. Cette dépendance est indiquée en traçant un demi-cercle, appelé un arc ET, à travers les arcs $(u, v_1), (u, v_2), \dots, (u, v_k)$. Un tel arc ET a un coût associé, qui est le coût encouru lors du calcul de u à partir de $v_1, v_2, \dots, v_k$. Par exemple, la figure (1.5)-(a) montre un ET-DAG. La vue de nœud v_1 est calculée à partir d'un ensemble de vues $\{v_2, v_3, v_4\}$.

Pour une vue ou une requête v de ET-OU DAG, avec v comme source et les relations de base comme puits. Chaque vue non-puits est associée à un ou plusieurs arcs ET. La définition (1) fournit d'une manière formelle la définition de graphe de vues ET OU-DAG. La figure (1.5)-(b) montre l'exemple d'une expression ET-OU DAG. La vue de nœud v_1 peut être calculée soit à partir de l'ensemble des vues $\{v_2, v_3\}$ ou $\{v_4, v_5\}$

Définition 1

Un graphe G-DAG avec des relations de base comme puits est appelé un graphe de vues ET-OU, pour un ensemble de vues ou requête $v_1, v_2, \dots, v_k$. Si pour chaque v_i, il y a un sous-graphe G_i dans G qui est une expression ET OU-DAG pour v_i. Chaque nœud v dans un graphe de vues ET-OU est associé aux paramètres suivants: espace de stockage, fréquence de requête (fréquence des requêtes sur v), fréquence de maintenance (fréquence des mises à jour sur v). La sélection de vue pour le problème de matérialisation à l'aide du graphe de vues ET-OU est définie par[3] :

Définition 2

Étant donné un graphe de vues ET-OU-DAG et une quantité S (espace de stockage), le problème de sélection des vues consiste à sélectionner un ensemble de vues M qui constituent un sous-ensemble des nœuds de G, cela minimise le temps total de réponse de la requête, sous la contrainte que l'espace total occupé par M est inférieur à S sous une contrainte de coût de maintenance.

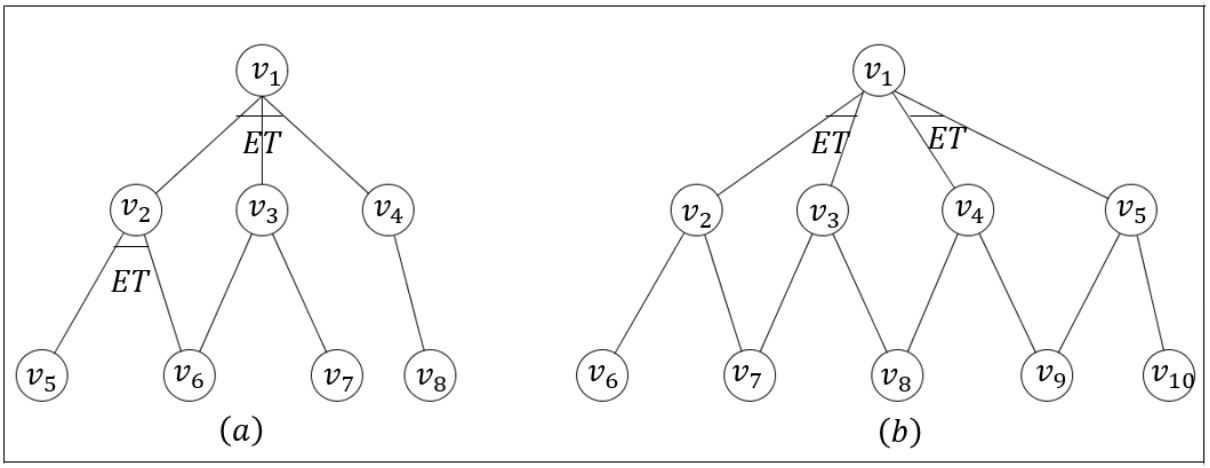

Figure 1.5 (a) Expression ET, (b) Expression ET/OU

Cette représentation est largement utilisée pour le problème général de la sélection des vues dans un entrepôt de données. Le graphe de vues ET-OU DAG, représente le scénario général d'entrepôt de données d'une manière facilement compréhensible pour analyser les requêtes et leurs vues composantes. Par conséquent, il est approprié pour calculer le coût des requêtes (en utilisant les ensembles de vues matérialisées dans le graphe) et le coût des maintenances. Pour chaque requête, ses vues et ses tables de base, est considérée individuellement et, par conséquent, le partage des vues matérialisées par plusieurs requêtes est ignoré.

1.4.3 Espace de recherche à base de plan multiple d'exécution des vues

Un plan ou un arbre de requête est utilisé pour représenter une requête, généralement cette requête est exprimée dans l'algèbre relationnelle. Cette représentation est équivalente à la requête d'origine. Un plan de requête est une structure de données qui correspond à une expression de l'algèbre relationnelle. Ce plan représente les relations d'entrée d'une requête en tant que nœuds feuilles de l'arbre et les opérations d'algèbre relationnelle en tant que nœuds internes.

Une exécution de l'arbre de requête consiste à exécuter une opération d'un nœud interne chaque fois que ses opérandes sont disponibles, en remplaçant plus tard ce nœud interne par la relation qui résulte de l'exécution de l'opération. L'exécution se termine lorsque le nœud racine est exécuté et produit la relation de résultat pour la requête. La représentation du plan de requête n'indique pas dans quel ordre les opérations doivent être effectuées en premier lieu. L'optimisation de plan multiple de requêtes doit montrer l'ordre des opérations pour l'exécution de la requête.

a) Expressions communes

Normalement, l'espace de recherche pour un problème de sélection des vues est construit en utilisant toutes les sous-expressions communes ou similaires parmi les requêtes. Le concept d'une sous-expression commune est initialement appelé expression identique ou équivalente, et plus tard, le terme inclut la notion de partage. Par la suite, cette notion de partage entre les requêtes a inclu trois catégories comme montré dans la figue (1.6).

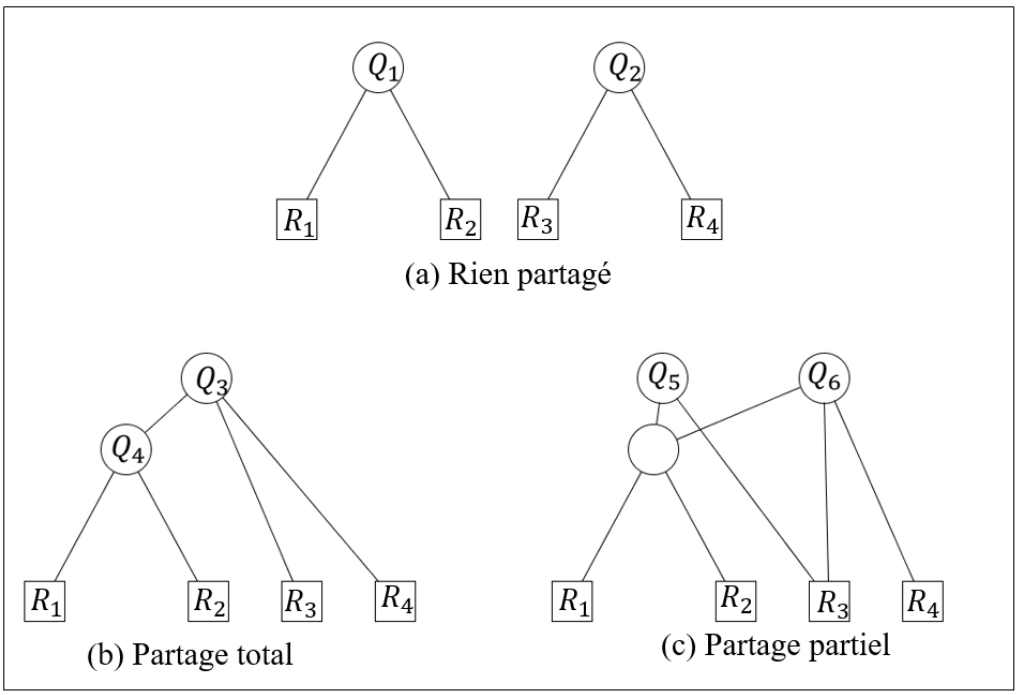

Figure 1.6 Trois catégories de partage

Il existe trois catégories de partage entre les requêtes : la première catégorie, qui est caractérisée par l'absence des sous-expressions de partage (la figure 1.6 (a)). La seconde catégorie est le partage total, comme indiqué à la même figure (b), la requête Q4 participe d'une manière total à l'exécution de la requête Q3. La troisième catégorie est le partage partiel où les deux requêtes Q5 et Q6 partagent une sous-expression, représentée à la figure (1.6)-(c). Une fois les sous-expressions communes détectées, elles sont exploitées pour construire le plan optimal équivalent pour plusieurs requêtes de plan de traitement.

b) Optimisation de l'espace de recherche

L'espace de recherche est obtenu en sélectionnant exactement un plan pour chaque requête et identifier les sous-expressions dans lesquelles le partage est possible. Les plans sélectionnés devraient minimiser le coût du plan d'exécution global[43]. Dans la deuxième étape, nous devons donc sélectionner un ensemble approprié de vues matérialisées de cet espace de recherche. L'algorithme pour générer l'espace de recherche est le suivant. D'abord, nous devons créer tous les plans d'exécution alternatifs pour chaque requête, puis, fusionner les plans de traitement de requête individuels pour générer un espace de recherche et finalement choisir le meilleur plan en terme du coût. Au sens que le coût du plan d'exécution global soit minimal.

L'algorithme suivant décrit la procédure à suivre pour générer le plan optimal, et cet algorithme est schématisé dans la figure (1.7).

Algorithme pour construire l'espace de recherche optimal

1. Pour chaque requête q_i, générer les plans possibles de traitement de chaque requête P_{ik_i}
2. Créer une liste des plans $P = \langle P_{1k_1}, P_{2k_2}, \dots, P_{nk_n} \rangle$,
3. pour $i = 1$ à n faire

   Choisir le plan optimal qui regroupe un plan de chaque requête,

4. Fin pour

Fin

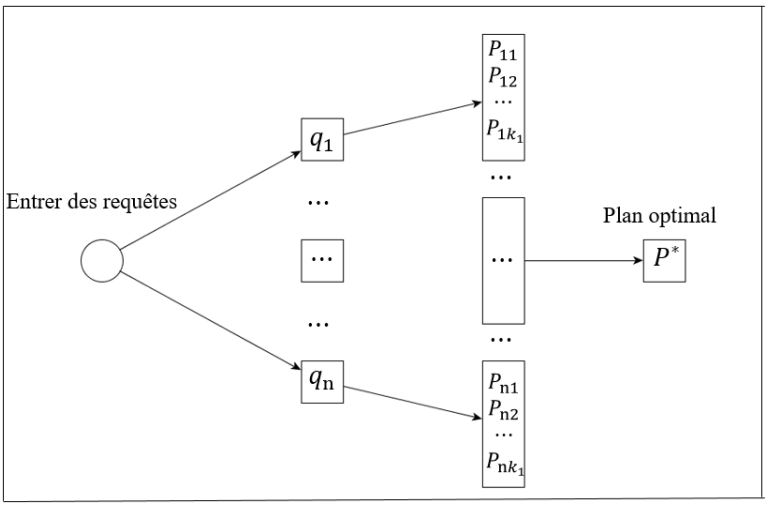

Figure 1.7 Présentation de l'optimisation des requêtes basée sur les coûts

A base de deux requêtes Q1 et Q2, l'espace de recherche est obtenu par la sélection d'un plan pour chaque requête, et le meilleur choix de sélection permet d'avoir un espace de recherche optimal qui est le résultat de la fusion de ces deux plans sélectionnés (figure 1.8).

Q1. *select c_nation, s_nation, d_year,*
sum(lo_revenue)
from customer, lineorder, supplier, dates
where lo_custkey = c_customerkey
and lo_suppkey = s_suppkey

and lo_orderdatekey = d_datekey

and c_region = 'ASIA'

and s_region = 'ASIA'

and d_year >= 1992 and d_year <= 1997

group by c_nation, s_nation, d_year ,

Q2. *select c_city, s_city, d_year,*

sum(lo_revenue)

from customer, lineorder, supplier, dates

where lo_custkey = c_customerkey

and lo_suppkey = s_suppkey

and lo_orderdatekey = d_datekey

and c_nation = 'UNITED STATES'

and s_region = 'ASIA'

and d_year >= 1992 and d_year <= 1997

group by c_city, s_city, d_year ,

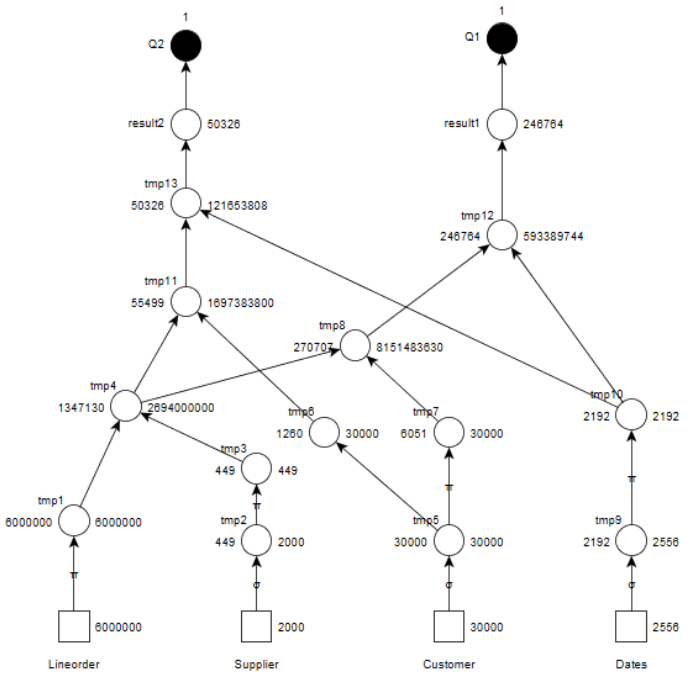

Figure 1.8 Espace de recherche construit par la fusion de deux requêtes

1.4.4 Espace de recherche à base de fouille de données

Une autre approche est conçue dans l'objectif de proposer automatiquement des solutions aux administrateurs d'entrepôt de données pour optimiser le temps d'accès aux données. Le principe de cette approche consiste à appliquer des techniques d'exploration de données sur une charge des requêtes (ensemble de requêtes) afin de déduire une configuration quasi-optimale de vues matérialisées. Cette approche, pour résoudre le problème de sélection de vue, consiste à trouver des sous-expressions communes correspondant à des résultats intermédiaires appropriés à la matérialisation. Dans l'espace de recherche à base de fouille de données, toutes les requêtes et les attributs sont identifiés et analysés qui sont issus d'une charge des requêtes pour construire une matrice binaire de requête vs attribut. Dans cette matrice, chaque ligne représente une requête et chaque colonne est un attribut. Une cellule est marquée par un (1) si un attribut particulier est présent dans une requête, et zéro (0) sinon. Des techniques d'exploration de données sont appliquées à cette matrice pour obtenir un ensemble de vues candidates à la matérialisation. Bien que la représentation de la matrice soit facile à implémenter et directement utilisable par les algorithmes de fouille de données, la principale difficulté est l'analyse syntaxique de la charge des requêtes. C'est parce que l'analyse à travers de nombreuses sous-requêtes et résultats intermédiaires pour générer la matrice binaire nécessite un algorithme infaillible[41].

1.5. Travaux sur le problème de sélection de vues

Les problèmes d'optimisation se posent dans presque tous les domaines ; du domaine de développement scientifique aux domaines d'application industrielle. Ils émergent quand la tâche est de trouver la meilleure de solutions possibles à un problème donné, à condition qu'il y ait une idée claire de la façon d'évaluer la qualité de la solution. Contrairement à d'autres problèmes d'optimisation, avoir un nombre fini de solutions candidates. Par exemple, une forme évidente de résolution consisterait à énumérer toutes les solutions possibles en les comparants. Malheureusement, pour la plupart de ces problèmes, cette approche n'est pas réalisable, car le nombre de solutions possibles est simplement exponentiel, comme dans le cas du problème de sélection de vues. C'est la raison pour laquelle les chercheurs ont concentré leurs efforts sur l'utilisation de l'heuristique pour résoudre les problèmes de complexité exponentielle. Une heuristique consiste en une technique inspirée par des processus intuitifs qui cherche une bonne solution à un coût de

calcul acceptable, sans toutefois pouvoir garantir la solution optimale, et garantir à quel point la solution est proche de la solution optimale. Le défi consiste à produire, en temps réduit, des solutions aussi proches que possible de la solution optimale. De nombreux efforts ont été faits dans ce sens et des heuristiques très efficaces et flexibles ont été développées pour divers problèmes. Les méta-heuristiques, qui sont des heuristiques universelles, peuvent être facilement adaptées pour résoudre divers problèmes. Parmi les algorithmes classés comme méta-heuristiques celles qui ont émergé au cours des dernières décennies, on peut distinguer: les algorithmes génétiques, réseaux neuronaux, algorithme d'escalade, recuit simulé, colonie de fourmis, etc. Un certain nombre d'œuvres sont également présentés sur le problème de sélection de vue.

Gupta et al. [9] ont proposé un algorithme glouton qui sélectionne la vue la plus bénéfique par unité d'espace disque à chaque étape et l'ajoute à l'ensemble des vues déjà matérialisées. La complexité temporelle de l'algorithme est $O(kn^2)$ où k est le nombre de vues matérialisées et n représente le nombre de toutes les vues candidates. Derakhshan et al. [38] ont introduit l'approche de recuit simulé dans la résolution du problème de sélection des vues. L'approche proposée apporte une amélioration significative de la qualité de la solution obtenue. Lee et al. [44] ont exploré l'approche évolutionniste pour résoudre le problème de sélection de la vue où ils ont modélisé l'espace de recherche en tant que graphe ET/OU. Talebi et al. [45] ont modélisé le problème de sélection de vue comme une programmation linéaire en nombres entiers (PLNE). Le modèle PLNE a été utilisé pour obtenir les solutions optimales garanties. De plus, une méthode heuristique a été proposée afin de trouver la solution inexacte compétitive dans les cas où une méthode exacte n'est pas applicable. Ashadevi et al. [46] ont développé un cadre pour la sélection des vues matérialisées afin d'obtenir la meilleure combinaison du temps de réponse aux requêtes et du temps de mise à jour soumis à une contrainte d'espace disque. Les vues avec une fréquence de requête élevée sont sélectionnées en tant que vues matérialisées initiales. Aucune comparaison avec des travaux similaires n'a été effectuée. Li et al. [47] ont introduit le modèle de programmation linéaire en nombres entiers afin d'obtenir une solution optimale globale. Le résultat expérimental montre l'efficacité de l'approche proposée dans les instances. Aouiche et al. [41] ont exploité une technique d'exploration de données, afin de déterminer des clusters de requêtes similaires. Bauer et al. [48] se sont concentrés sur le problème de la sélection des vues dans le contexte des architectures d'entrepôt de données distribuées. Mami et al. [49] ont modélisé le problème de sélection des vues comme problème de satisfaction des

contraintes et ont appliqué l'approche de programmation par contraintes pour résoudre le problème. Lawrence et al. [50] ont exploré la sélection dynamique de vue dans laquelle la distribution des requêtes change avec le temps, et un sous-ensemble d'un ensemble de vues matérialisées est mis à jour pour mieux servir les requêtes entrantes. Shah et al. [51] ont proposé une approche hybride pour la sélection des vues matérialisées. L'idée est de partitionner la collection de toutes les vues en un ensemble statique et dynamique de sorte que les vues sélectionnées pour la matérialisation de l'ensemble statique soient persistantes sur plusieurs fenêtres de requête, alors que les vues sélectionnées de l'ensemble dynamique peuvent être interrogées (remplacées ou supprimées).

1.6 Conclusion

Dans ce chapitre, nous avons évoqué une architecture d'entrepôt de données et sa modélisation multidimensionnelle. Ensuite, nous avons introduit les principales techniques d'optimisation des requêtes pour améliorer la performance de l'exécution de celles-ci. Quatre espaces de recherche, à base de cube en treillis multidimensionnel, à base de graphe des vues ET-OU, à base de plan multiple d'exécution des vues et à base de fouille de données ont été exposés dans ce chapitre pour le problème de sélection des vues. Nous nous sommes limités dans notre étude à l'espace de recherche à base de plan multiple d'exécution de vues et nous avons adopté la technique de matérialisation des vues pour améliorer la performance des requêtes. Nous avons aussi enchainé par un état d'art dans le possible sur des travaux réalisés pour résoudre le problème de sélection des vues et qui ont proposé différentes approches suivant différents espaces de recherche.

Chapitre 2 : Réseaux de neurones récursifs de Hopfield et approche évolutionniste

2.1. Introduction

L'intelligence artificielle est devenue un outil d'analyse incontournable dans les domaines les plus variés de la physique et des mathématiques appliquées. Cette discipline est riche au niveau des techniques sophistiquées pour analyser divers problèmes d'optimisation de différents domaines. Parmi ces techniques, nous citons les réseaux de neurones récurrents de Hopfield et les algorithmes génétiques, ces deux techniques font l'objet de ce chapitre.

Les réseaux de neurones récurrents de Hopfield sont communément connus sous le nom de réseau de Hopfield parce que l'apparition de ces derniers a fourni un grand coup de pouce aux réseaux connexionnistes[52]. Les connexions, de réseaux récurrents, relient les sorties aux entrées, ce qui fournit une grande généralité dans son fonctionnement car la sortie du réseau à un certain moment dépendra non seulement des entrées courantes et des poids de connexion, mais aussi des sorties offertes par le réseau dans les moments précédents. Cette caractéristique fait que les réseaux récurrents présentent des propriétés similaires à la mémoire humaine, dans laquelle l'état des sorties des neurones du cerveau dépend, en partie, des entrées précédentes.

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Ils permettent une simplification dans la formulation et la résolution de problèmes d'optimisation en incorporant une solution potentielle à un problème spécifique dans une structure du chromosome tout en appliquant des opérateurs de sélection et de croisement à ces structures afin de préserver les informations critiques du problème[53].

Ce chapitre comporte deux parties. Dans la première partie, nous présentons le réseau de de Hopfield, sa structure et son fonctionnement, aussi son évolution vers un attracteur. Nous présentons aussi le réseau de Hopfield continu et sa topologie. Dans la deuxième partie, nous

présentons les algorithmes génétiques comme une autre technique inspirée de la biologie et qui a reçu beaucoup d'attention dernièrement. Nous décrivons toutes les opérations des algorithmes génétiques, représentation génétique, création de population, fonction d'adaptation, la sélection, opérations de reproductions et la convergence de cet algorithme.

2.2 Réseau de Hopfield

Du perceptron simple au perceptron multicouche, les connexions sont toujours à sens unique. On va de l'entrée à la sortie et l'information sortante n'est jamais recyclée. La topologie feedforward des réseaux non bouclés n'est que l'une des topologies possibles. Cependant, il existe des réseaux récurrents où les signaux de sortie sont recyclés en entrée [54], [55].

En 1982, J.J.Hopfield [56], physicien de l'institut de technologie de Californie, a publié un article sur le développement des réseaux neuronaux, proposant un réseau de neurones récurent monocouche, appelé par la suite le réseau de Hopfield.

J .J . Hopfield a introduit le concept de la fonction énergétique dans le réseau de neurones récurrent. Ce concept est d'une grande importance pour l'étude des réseaux de neurones, il fait reposer les jugements de la stabilité du réseau neuronal sur une base fiable. En 1985, Hopfield avec D.W. Tank [57] ont également mis en place le réseau de Hopfield avec des circuits électroniques analogiques. Ils ont réussi à résoudre le problème du voyageur de commerce, en ouvrant ainsi un réseau neuronal capable de faire un traitement intelligent de l'information, et une nouvelle voie prometteuse pour le traitement des problèmes d'optimisation via les réseaux de neurones.

Contrairement aux réseaux feedforward qui généralement ne tiennent pas compte du lien entre la sortie et l'entrée des neurones, le réseau de Hopfield tient compte du lien entre la sortie et l'entrée des neurones grâce à sa dynamique qui est caractérisée par une équation différentielle. Dans ce contexte, nous proposons plusieurs modélisations de cette équation pour construire un nouveau modèle mathématique de ce type de réseau.

Les poids de ce réseau de Hopfield ne sont pas obtenus par l'apprentissage, mais calculés d'avance selon certaines règles. Les poids ne changeront pas une fois déterminés, et l'état de chaque neurone sera constamment mis à jour jusqu'à l'évolution de réseau vers un état stable. Dans le cas de la stabilité, l'état de chaque neurone est la solution du problème. Le réseau de Hopfield est composé en deux types de modèles : réseau de Hopfield discret (discrete Hopfield neural network) et réseau de Hopfield continu (continuous Hopfield neural network).

2.2.1 Structure et mode de fonctionnement du réseau de Hopfield discret

La structure du réseau de Hopfield discret est illustrée dans la figure (2.1). Il s'agit d'un réseau récurrent composé d'une seule couche avec n neurones. Dans ce réseau, la sortie de n'importe quel neurone x_i est renvoyée comme entrée à tous les neurones x_j à travers la connexion w_{ij}. En d'autre terme, chaque neurone reçoit les informations renvoyées par tous les neurones à travers les connexions des poids, son but est de permettre à la sortie de n'importe quel neurone d'être contrôlée par la sortie de tous les neurones, de sorte que la sortie de chaque neurone peut se restreindre. Chaque neurone a un seuil I_i pour refléter le contrôle du bruit d'entrée.

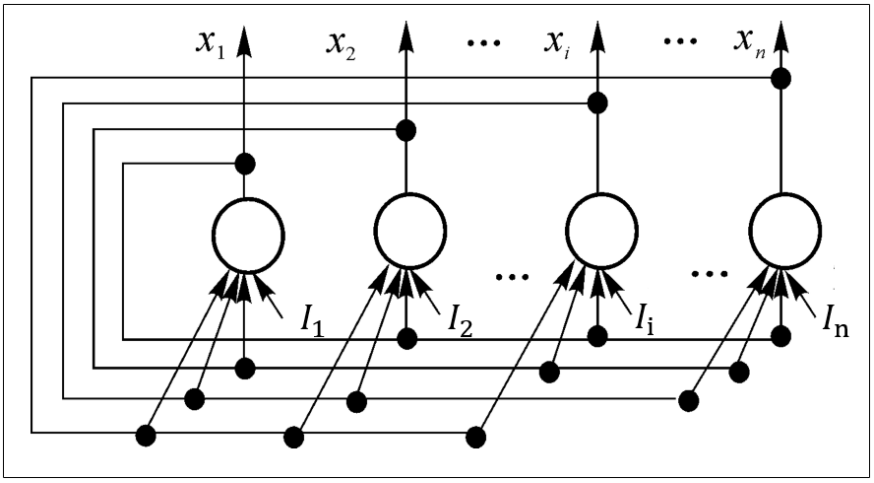

Figure 2.1 Structure du réseau de Hopfield discret

- Etats du réseau

La sortie de chaque neurone du réseau est obtenue par une fonction de transfert, cette sortie est appelée état, notée par x_i, et l'ensemble de tous les états neuronaux constitue l'état défini par $x(t) = \big(x_1(t), x_2(t), \dots, x_n(t)\big)$. L'état initial du réseau récurrent de Hopfield est exprimé par :

$$x(0) = \big(x_1(0), x_2(0), \dots, x_n(0)\big)$$

Le processus dynamique de l'évolution de neurones permet à mettre à jour l'état de chaque neurone par la formule suivante :

$$x_i = f(u_i)\ i = 1,2, \dots, n$$

Dans ce cas, $f(.)$ est une fonction de transfert qui est souvent utilisée comme une fonction de seuil.

$$x_i = sgn(u_i) = \begin{cases} 1\ si\ u_i \geq 0 \\ -1\ si\ u_i < 0 \end{cases} \quad \forall i = 1,2,\dots,n \tag{2.1}$$

L'entrée nette est :

$$u_i = \sum_{j=1}^{n} \left(w_{ij}.x_j\right) - I_i\ ,\ \forall\ i = 1,2,\dots,n \tag{2.2}$$

En ce qui concerne les poids, le réseau de Hopfield est caractérisé généralement par $w_{ii} = 0\ et\ w_{ij} = w_{ji}$. Dans l'état stable du réseau de Hopfield, la sortie de chaque neurone ne changera plus. A ce moment, on dit que la sortie du réseau est atteinte, et exprimée par :

$$\lim_{t\to\infty} X(t)$$

- Modes de mise à jour

La mise à jour de réseau de Hopfield est souvent exprimée par deux manières, soit en mode asynchrone, soit en mode synchrone.

- Mode asynchrone

A chaque instant, un seul neurone est mis à jour et ajusté. Ce neurone peut être sélectionné au hasard ou suivi par un ordre préétabli. Le résultat de cet ajustement influence sur l'entrée du neurone suivant. A la mise à jour du j^{ème} neurone, on définit l'état du réseau de Hopfield par :

$$x_i(t+1) = \begin{cases} sgn\left(u_i(t)\right)\ si\ i = j \\ x_i(t)\ si\ i \neq j \end{cases} \tag{2.3}$$

- Mode synchrone

Dans le mode synchrone ou mode parallèle tous neurones mettent à jour leurs états en même temps.

$$x_i(t+1) = sgn\left(u_i(t)\right)\ \forall i = 1,2,\dots,n \tag{2.4}$$

2.2.2 Stabilité du réseau et attracteur

Le réseau de neurones de Hopfield est un réseau qui stocke un certain nombre d'états de stabilité prédéfinis. Lors de l'exécution, le réseau renvoie sa sortie comme entrée suivante lorsqu'elle agit sur le réseau en tant que mode d'entrée initial. Après plusieurs itérations, le

réseau se stabilisera éventuellement à un état de stabilité prédéfini, à condition que la structure du réseau réponde à une certaine condition.

Soit $X(0)$ le vecteur d'activation initial du réseau, qui agit sur le réseau uniquement à l'instant initial $t = 0$. Après la mise à jour de $X(0)$, le réseau remplacera ce vecteur par l'entré suivante $X(1)$ à l'instant $t = 1$. En tant que système dynamique non linéaire, le réseau de neurones de Hopfield a des caractéristiques dynamiques telles que la stabilité et l'instabilité de ses états.

- **Stabilité du réseau**

Selon l'analyse de l'état de fonctionnement du réseau, le réseau de Hopfield est essentiellement un système dynamique non linéaire. A partir de l'état initial $X(0)$ si le réseau de Hopfield converge après un nombre fini d'itérations, alors son état ne change pas, c'est-à-dire $X(t+1) = X(t)$. Donc le réseau est dit stable. Dans le cas de convergence, le réseau de Hopfield peut converger de n'importe quel état initial vers un état stable comme indiqué dans la Figure (2.2)-(a). Dans le cas de l'instabilité de réseau de Hopfield, le réseau diverge. Cette divergence peut être interprétée par l'oscillation auto-entretenue de la limite, appelée réseau en anneau fini. La figure (2.2)-(b) donne son diagramme de phase. Si la trajectoire de l'état du réseau change dans une certaine plage, mais ni répète ni arrête, l'état change en un nombre infini de trajectoires, ce phénomène est appelé chaos, le diagramme de phase montré dans la figure (2.2)-(c) ci-dessous.

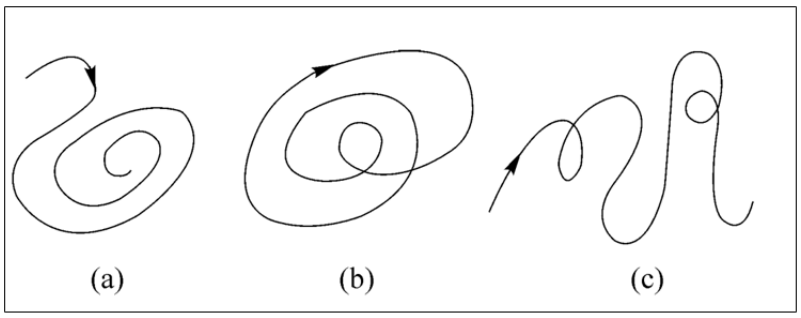

Figure 2.2 Trois types de stabilité d'un réseau : (a) stabilité (b) oscillation (c) chaos

- **Attracteur et fonction d'énergie**

Lorsque le réseau atteint un état stable, là on parle de l'attracteur du réseau. Le comportement final d'un système dynamique est déterminé par son attracteur, et l'existence d'attracteurs constitue la base de la mémoire des informations stockées. Si nous considérons l'attracteur

comme la solution du problème, alors l'évolution de l'état initial vers l'attracteur est le processus de résolution de ce problème. La définition et le théorème de l'attracteur de réseau en mode asynchrone sont donnés ci-dessous.

Définition 2.1 : Si l'état X du réseau vérifie $X = f(WX - I)$, alors X est l'attracteur du réseau.

Théorème 2.1 : Pour un réseau en mode asynchrone, si l'état du réseau est réglé de manière asynchrone et que la matrice de poids de connexion W est une matrice symétrique, le réseau converge finalement vers un attracteur pour tout état initial[57].

Dans ce qui suit, le théorème (2.1) est prouvé en analysant la fonction d'énergie. On définit la fonction énergétique du réseau comme:

$$E(t) = -\frac{1}{2}X^T(t)WX(t) + X^T(t)I \tag{2.5}$$

La variation de l'énergie du réseau est notée par ΔE et la variation d'état du réseau est notée par ΔX, alors :

$$\Delta E(t) = E(t+1) - E(t) \tag{2.6}$$

$$\Delta X(t) = X(t+1) - X(t) \tag{2.7}$$

En substituant (2.5) et (2.7) dans (2.6), la quantité de changement dans l'énergie du réseau peut être étendue à :

$$\Delta E(t) = E(t+1) - E(t)$$

$$= -\frac{1}{2}[X(t) + \Delta X(t)]^T W[X(t) + \Delta X(t)] + [X(t) + \Delta X(t)]^T I - \left[-\frac{1}{2}X^T(t)WX(t) + X^T(t)I\right]$$

$$= -\Delta X^T(t)WX(t) - \frac{1}{2}\Delta X^T(t)W\Delta X(t) + \Delta X^T(t)I$$

$$= -\Delta X^T(t)[WX(t) - I] - \frac{1}{2}\Delta X^T(t)W\Delta X(t) \tag{2.8}$$

Le théorème (2.1) fournit une mise à jour de chaque neurone d'une manière asynchrone, il n'y a qu'un seul état d'un neurone est ajusté à l'instant t, soit le neurone i concerné par la mise à jour, et que $X(t) = (0, \dots, 0, \Delta x_i(t), 0, \dots, 0)^T$ et considérant que W est une matrice symétrique, alors:

$$\Delta E(t) = -\Delta x_i(t)\left[\sum_{j=1}^{n}\left(w_{ij}.x_j - I_i\right)\right] - \frac{1}{2}\Delta x_i^2(t)w_{ii}$$

Pour $w_{ii} = 0$, et en introduisant (2.2) à la formule ci-dessus qui peut être simplifiée :

$$\Delta E(t) = -\Delta x_i(t).u_i(t) \qquad (2.9)$$

Considérons toutes les situations qui peuvent survenir dans l'équation ci-dessus.

- Cas A : $x_i(t) = -1\ et\ x_i(t+1) = 1$. À partir de l'équation (2.7), nous obtenons $\Delta x_i(t) = 2\ et\ u_i(t) \geq 0$ qui peut être obtenu à partir de l'équation (2.1), et en substituant en (2.9), nous obtenons $\Delta E(t) \leq 0$.
- Cas B : $x_i(t) = 1\ et\ x_i(t+1) = -1$ donc $\Delta x_i(t) = -2\ et\ u_i(t) \leq 0$ et en substituant en (2.9), nous obtenons $\Delta E(t) \leq 0$.
- Cas C : $x_i(t) = x_i(t+1)$ donc $\Delta x_i(t) = 0$ et en substituant en (2.9), nous obtenons $\Delta E(t) = 0$.

Tous les trois cas ci-dessus incluent les cas possibles de l'équation (2.9), à partir de laquelle nous pouvons voir que dans tous les cas $\Delta E(t) \leq 0$. En d'autre terme, lors de l'évolution dynamique du réseau, l'énergie est toujours décroissante ou reste-la même.

L'analyse de la convergence de $E(t)$ vers une constante correspond à l'état stationnaire du réseau. Lorsque $E(t)$ converge vers une constante $\Delta E(t) = 0$. Ce qui correspond aux deux cas suivants:

- Cas A: $x_i(t) = x_i(t+1) = 1\ ou\ x_i(t) = x_i(t+1) = -1$. Dans ce cas, l'état du neurone i ne change plus son état, indiquant que le réseau est entré en régime permanent, l'état du réseau correspondant est l'attracteur du réseau.
- Cas B: $x_i(t) = -1\ , x_i(t+1) = 1\ et\ u_i(t) = 0$. Dans ce cas, le réseau continuera à évoluer, ce qui contredit le fait que $E(t)$ converge vers une constante.

Théorème 2.2 : Pour le réseau en mode synchrone, si l'état est réglé de manière synchrone et que la matrice de poids de connexion W est une matrice symétrique définie non négative, le réseau converge finalement vers un attracteur pour tout état initial[58].

La comparaison entre le théorème (2.1) et théorème (2.2), nous révèle que lorsque le réseau fonctionne en mode synchrone, la matrice de poids W est plus exigeante, Si W ne peut pas

répondre à l'exigence de matrice symétrique définie non négative, le réseau apparaîtra comme une oscillation auto-entretenue. Les méthodes asynchrones ont une meilleure stabilité que les méthodes synchrones mais avec l'inconvénient est qu'elles perdent l'avantage de traitement parallèle.

L'analyse de ci-dessus montre que dans le réseau de l'état initial au processus d'évolution stationnaire, l'énergie du réseau a toujours été de réduire la direction de l'évolution, lorsque l'énergie s'est finalement stabilisée, la constante correspond à l'état minimum de l'énergie du réseau. Le plus petit état est le puits énergétique du réseau, et l'énergie correspond bien à l'attracteur du réseau.

Exemple 2.1, pour un réseau en mode asynchrone à 3 neurones, qui est représenté par un graphe non orienté comme le montre la figure (2.3), Les poids et les seuils sont indiqués sur la même figure pour calculer l'état de l'évolution du réseau.

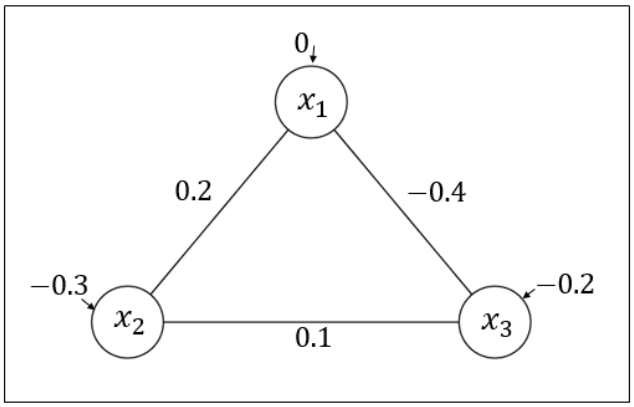

Figure 2.3 Réseau de trois neurones

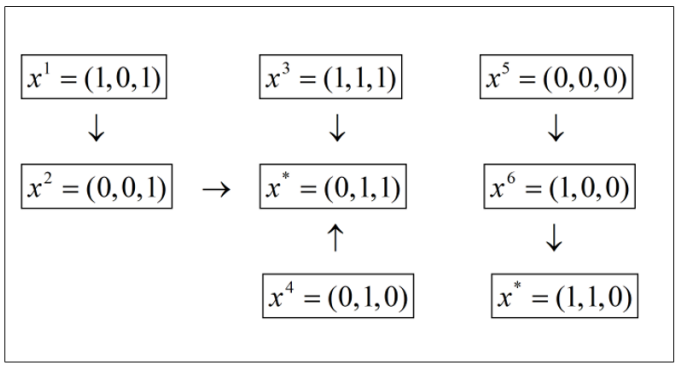

Figure 2.4 Diagramme d'évolution de réseau de trois neurones

Le réseau de la figure (2.3) doit avoir 8 type d'état de $x = (x_1, x_2, x_3)$. Nous définissons l'état initial du réseau récurrent de Hopfield par $x^1 = (0,0,0)$ en suivant l'ordre de mise à jour de $1 \rightarrow 2 \rightarrow 3$.

- Étape 1: Mise à jour de $x_1, x_1 = sgn(0 \times 0.2 + 0 \times (-0.4) - 0) = sgn(0) = 1$ et l'état des autres neurones ne change pas. Alors l'état du réseau passe de $(0,0,0)$ à $(1,0,0)$.
- Étape 2: Maintenant, à base de l'état du réseau précèdent défini par $(1,0,0)$, nous pouvons mettre à jour l'état de neurone x_2 par la relation $x_2 = sgn(1 \times 0.2 + 0 \times 0.1 + 0.3) = sgn(0.5) = 1$ et les autres neurones restent inchangés. Donc l'état du réseau devient $(1,1,0)$.
- Étape 3: La mise à jour de neurone x_3 est obtenu par $x_3 = sgn(1 \times (-0.4) + 1 \times 0.1 + 0.2) = sgn(-0.1) = 0$, à ce stade, l'état du réseau est $(1,1,0)$.

La même procédure peut être faite pour l'évolution des autres états. La figure (2.4) représente les huit états possibles d'évolution. Sur la même figure les $x^* = (1,1,0)$ et $x^*(0,1,1)$ sont les attracteurs du réseau. À partir de n'importe quel état, le réseau atteindra un de ces deux états stables après plusieurs mises à jour.

2.2.3 Conception des poids de réseau

La distribution des attracteurs est déterminée par les poids du réseau (y compris le seuil), et le cœur de la conception des attracteurs est de savoir comment concevoir un ensemble de poids appropriés. Pour que les poids conçus répondent aux exigences, la matrice de poids doit répondre aux exigences suivantes:

- W doit être une matrice symétrique afin d'assurer la convergence du réseau lorsque l'on travaille en mode asynchrone,
- W doit être une matrice symétrique définie non négative afin d'assurer la convergence du réseau lorsque l'on travaille en mode synchrone,
- Pour s'assurer qu'un échantillon donné est un attracteur de réseau.

Selon le nombre d'attracteurs nécessaire pour une application, la méthode suivante peut être utilisée.

- **Méthode de conception**

Cette méthode de conception consiste principalement à faire correspondre l'échantillon de motif stocké à la valeur minimale de la fonction d'énergie du réseau. Par exemple, s'il y a m modèles de mémoire à n dimensions, nous voulons concevoir les poids de connexion w_{ij} de réseau et les seuils I_i de sorte que ces m modèles soient exactement m minima pour la fonction d'énergie du réseau.
Étant donné un vecteur de m motifs :

$$X^k = [X_1^k, X_2^k, \dots, X_n^k]^T \; avec \; k = 1,2, \dots, m \; et \; X_i^k \in \{-1,1\} \; \forall i = 1,2, \dots, n$$

Le seuil de chaque neurone $I_i = 0$ et la matrice W de connexion du réseau est calculé par la formule suivante :

$$W = \sum_{k=1}^{m} X^k . (X^k)^T$$

Alors tous les vecteurs X^k $(1 \leq k \leq m)$ sont des puits de réseau. Par conséquent, la formule de la conception de poids est donnée par :

$$w_{ij} = \begin{cases} \sum_{k=1}^{m} x_i^k x_j^k & i \neq j \\ 0 & i = j \end{cases} \quad (\forall i, j = 1,2, \dots, n) \; (2.9)$$

2. 3 Réseau de Hopfield continu

2.3.1 Description de réseau de Hopfield continu

En 1984, Hopfield [58] a développé un nouveau modèle de Hopfield continu. La structure de base de réseau de neurones continu est similaire à celle de réseau de neurones discret sauf que tous les neurones fonctionnent de manière synchrone et la sortie de chaque neurone est un analogue qui varie continuellement avec le temps, ce qui rend le réseau continu meilleur que le réseau discret en terme de parallélisme et de performances en temps réel. Plus proche du mécanisme de travail du réseau de neurones biologiques. Le réseau de Hopfield continu peut être décrit par une équation différentielle à coefficients constants, mais il est plus visuel et intuitif de décrire avec un circuit électronique analogique, qui est facile à comprendre et facile à mettre en œuvre.

2.3.2 Architecture du réseau

A base du réseau de neurones récurrent discret, Hopfield avec Tank[57] a proposé le réseau de Hopfield continu. Dans ce réseau de neurones, l'entrée et la sortie du réseau sont analogues, et les neurones travaillent en mode parallèle.

La figure (2.5) présente un modèle de neurones de Hopfield continu. Sur la figure, la résistance R_{i0} et les capacités C_i sont connectées en parallèle pour simuler la caractéristique de retard des neurones biologiques, et la résistance R_{ij} simule les caractéristiques synaptiques. L'amplificateur opérationnel ici est un amplificateur non linéaire qui simule la caractéristique de saturation non linéaire (fonction d'activation en forme de sommation) de neurones biologiques, c'est-à-dire : $V_i = \frac{1}{1+e^{-U_i}}$

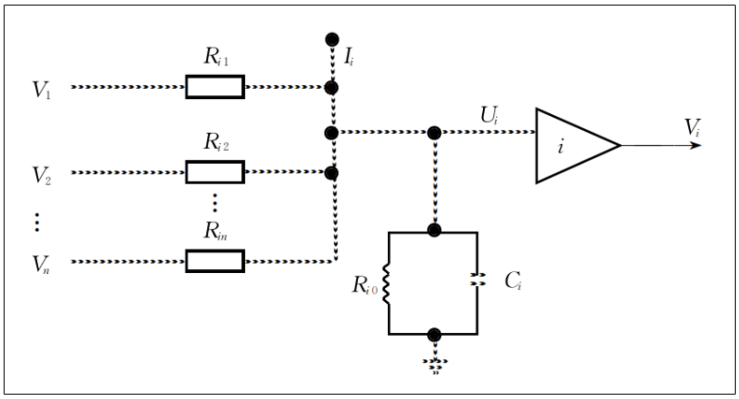

Figure 2.5 Modèle électronique du réseau de Hopfield continu

La structure du réseau de neurones de Hopfield continu est la même que celle du réseau discret: $W_{ij} = W_{ji}\ et\ W_{ii} = 0$. La fonction d'énergie est définie par :

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}x_i x_j + \sum_{i=1}^{n}\frac{1}{R_i}\int_0^{x_i} g^{-1}(t)dt - \sum_{i=1}^{n} x_i I_i \quad (2.10)$$

avec $g(x) = \frac{1}{1+e^{-x}}$

En général, le modèle continu est supérieur au modèle discret en termes du problème minimum local, en raison de sa surface lisse de l'énergie. De plus, le réseau continu avec un gain variable permet d'éviter les mauvais optimums locaux. Il est plus rapide et réel en comparaison avec un réseau discret[56]. Par conséquent, le modèle continu a dominé les techniques de résolution de problèmes d'optimisation, en particulier pour les problèmes combinatoires[57].

2.3.3 Optimisation quadratique via le réseau de Hopfield continu

En 1985, John Hopfield a construit une équipe dans le but d'étendre les applications de son modèle pour y inclure la résolution des problèmes d'optimisation[57]. Cette équipe a montré que les réseaux de neurones avec cette organisation de base pourraient être utilisés pour calculer des solutions à des problèmes d'optimisation spécifiques. Utilisation de la méthode proposée par Hopfield et Tank, la fonction d'énergie de réseau de Hopfield est équivalente à la fonction objective du problème d'optimisation qui doit être minimisée, tandis que les contraintes du problème sont incluses dans la fonction d'énergie sous le contrôle des termes de pénalité. Considérons le problème d'optimisation :

$$P = \begin{cases} Min\ f(x) \\ sc \\ [A]_1 x = b_1 \\ [A]_2 x = b_2 \\ \cdots \\ [A]_m x = b_m \end{cases}$$

Où $[A]_i$ (la i-ieme ligne de la matrice de contraintes A) et x est un vecteur de dimension n, et m est le nombre de contraintes. Ensuite, la fonction d'énergie est définie par

$$E(x) = \alpha f(x) + \beta_1([A]_1 x - b_1) + \beta_2([A]_2 x - b_2) + \cdots + \beta_m([A]_m x - b_m)$$

$\alpha, \beta_1, \cdots, \beta_m$ sont des paramètres de pénalité qui sont choisis pour refléter l'importance relative de chaque terme dans la fonction d'énergie.

Une fois la fonction d'énergie a été choisie, les paramètres du réseau (poids et des seuils) peuvent être déduits via une comparaison entre la fonction d'énergie donnée par l'équation (2.10) et la fonction d'énergie associée au problème traité. Les pondérations du réseau de Hopfield, W_{ij}, sont les coefficients des termes quadratique $x_i x_j$ de l'équation, et les seuils externes, I_i, sont les coefficients des termes linéaires x_i dans la fonction d'énergie choisie[52]. De toute évidence, un minimum du problème d'optimisation permettra également d'optimiser la fonction d'énergie, parce que la fonction objective $f(x)$, sera réduite au minimum, et la satisfaction de contraintes implique que les termes de pénalité seront égaux à zéro. Ce cas correspond à un état stable du réseau de Hopfield continu. Cependant, Les états stables du réseau de Hopfield continu ne correspondent pas nécessairement à des solutions possibles au problème d'optimisation, et c'est l'un des lacunes majeures de la formulation de Hopfield. Parce que la fonction d'énergie contient plusieurs termes, dont chacun est en compétition pour

être minimisé. De plus, une solution irréalisable pour le problème se posera lorsqu'au moins un des termes de pénalité de contrainte est non nul. Sinon, toutes les contraintes peuvent être satisfaites, mais un minimum local peut être rencontré qui ne minimise pas la fonction objective d'une manière globale. Dans ce cas, la solution est réalisable, mais pas 'bon'. Certes, un paramètre de pénalité peut être augmenté pour forcer son terme associé dans le but d'être réduits au minimum[59].

La solution à ce problème de compromis est de trouver les valeurs optimales des paramètres de pénalité qui équilibrent les termes de la fonction d'énergie et de veiller à ce que chaque terme est réduit au minimum avec la même priorité.

Le réseau de Hopfield continu est utilisé en programmation quadratique dans laquelle la recherche de l'extremum se ramène à la minimisation d'une fonction d'énergie $E(x)$ qui prend en considération le coût du problème et les contraintes auxquelles il doit obéir.

$$E(x) = E^O(x) + E^C(x)\ \forall x \in [0,1]^n$$

Où :

– $E^O(x)$ est directement proportionnel à la fonction objective.

– $E^C(x)$ est une fonction quadratique qui pénalise non seulement les contraintes violées du problème, mais garantit aussi la faisabilité de la solution obtenue par le réseau de Hopfield continu.

2.4 Algorithmes évolutionnistes

Les algorithmes évolutionnaires sont des techniques d'optimisation stochastiques qui adoptent les modèles de calcul inspirés par les processus de l'évolution naturelle tel que la sélection et reproduction. L'évolution naturelle peut être considérée comme un processus d'optimisation qui tend à améliorer la capacité de survie d'un système biologique dans un environnement compétitif soumis à des mutations continues. Selon la théorie de Darwin[60], dans un monde avec des ressources limitées, chaque individu est en concurrence avec les autres pour la survie. Les individus ayant de meilleures caractéristiques environnementales sont plus susceptibles de survivre et de se reproduire. De cette façon, les meilleures caractéristiques sont transmises aux générations suivantes, et au fil du temps, elles tendent à devenir dominantes parmi les individus de la population.

Parmi les algorithmes évolutionnaires, les algorithmes génétiques ont suscité l'intention de plusieurs chercheurs, vue son efficacité et sa simplicité d'adaptation à n'importe quel type de problème.

2.4.1 Algorithmes génétiques

Les algorithmes génétiques[53], [61], [62](AG) sont des techniques adaptatives utilisées pour résoudre les problèmes d'optimisation. Ils sont basés sur les processus génétiques menés par les organismes biologiques. Les AG ont été développés dans les années soixante-dix par John Holland et ses étudiants à l'université du Michigan[62]. Son objectif était de simuler le processus adaptatif des systèmes naturels et de développer des systèmes artificiels qui capturent les caractéristiques des systèmes naturels. Ce processus est traduit par un ensemble d'action présenté dans la figure (2.6). Au sens de comparaison, il y a trois caractéristiques principales qui distinguent l'AG des autres techniques de recherche et d'optimisation [63]:

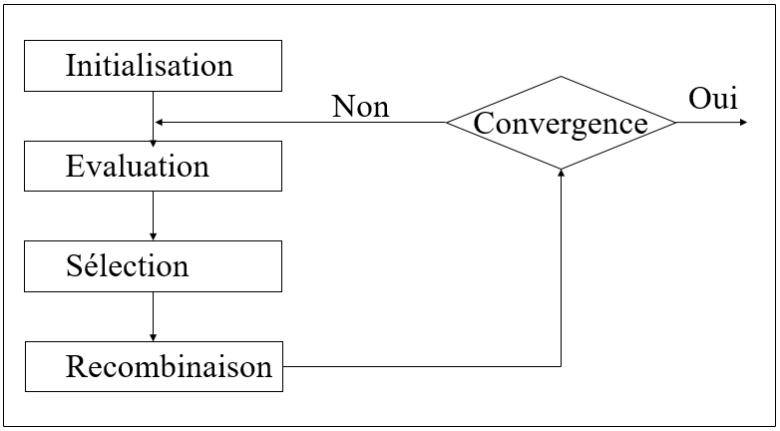

Figure 2.6 Schéma d'action de l'algorithme génétique

- Ils ne travaillent pas directement sur les paramètres à optimiser, mais sur une codification de ceux-ci.
- Ils ne recherchent pas une seule solution candidate, mais ils étudient des ensembles ou des populations de solutions candidates possibles.
- Ils utilisent directement la fonction objective, et non sa dérivée ou d'autres connaissances auxiliaires.

Les AG ont été utilisés avec succès dans des domaines très variés. Ceci est dû à diverses raisons. En premier lieu, malgré leur simplicité de calcul, ils sont extrêmement efficaces et efficients[63]. En outre, ils ne sont pas soumis à des restrictions sur l'espace de recherche (continuité, dérivabilité, etc.). Ils sont capables d'exploiter les informations disponibles sur un

espace de recherche initial inconnu, afin de mener la recherche vers des sous-espaces utiles. Cette fonctionnalité est particulièrement utile face à des espaces de recherche importants, complexes et peu connus, où les outils de recherche classiques (énumératifs, heuristiques, etc.) sont inappropriés, offrir une approche valable aux problèmes qui nécessitent des techniques de recherche efficaces et efficientes[63].

La forme canonique d'un AG encode chaque solution candidate à un problème donné sous la forme d'une chaîne binaire ou numérique (entière ou réelle), appelée chromosome. Les AG simulent l'évolution génétique (chromosomes) d'une population d'individus en utilisant la recombinaison (croisement et mutation) et la sélection. Dans un AG de base, l'opérateur de croisement échange des informations génétiques entre deux parents, afin d'obtenir de meilleurs enfants. En revanche, l'opérateur de mutation modifie aléatoirement la valeur de certaines des caractéristiques codées dans le chromosome, afin de préserver la diversité génétique de la population et d'éviter la convergence vers les maxima locaux. Chaque individu est évalué en lui attribuant un certain score d'aptitude ou de qualité atteint (fitness) qui est calculé proportionnellement à la valeur obtenue par l'individu lors de l'évaluation de la fonction objectif. La génération suivante est formée avec les individus qui ont le meilleur score à travers les opérateurs de sélection. Ce qui suit détaillera les aspects de la codification et les opérateurs les plus couramment utilisés par l'AG, afin d'offrir une vision plus spécifique des avantages et des caractéristiques offertes par l'AG lorsqu'il est utilisé dans des tâches d'optimisation[64].

2.4.2 Représentation génétique

Pour appliquer l'AG à un problème spécifique, nous devons d'abord définir une représentation génétique appropriée pour la solution. Nous devons trouver un moyen de coder toute solution pour un problème dans un chromosome avec une certaine structure. Cette structure partagée par tous les chromosomes est appelée représentation génétique[65], [66].

Généralement, les solutions sont codées dans une chaîne de bits d'une longueur donnée. Les chromosomes des chaînes de bits sont très pratiques car ils peuvent présenter une solution de n’importe quel type de problème. Historiquement, les AG ont essayé d'être universels, capables de gérer un large éventail de problèmes. Par conséquent, le codage binaire était considéré comme une représentation standard pouvant convenir à presque n'importe quel type d'espace de recherche. En fait, une chaîne de caractères peut coder des entiers, des réels, des ensembles ou tout ce qui est approprié pour le problème. De plus, les chromosomes binaires

sont très faciles à manipuler. La mutation et la recombinaison des chaînes de bits peuvent être effectuées avec des opérateurs très simples[64].

Il y a des points importants qu'une représentation génétique devrait essayer de respecter. Chaque gène, ou élément de base du chromosome, doit correspondre à une caractéristique particulière. Dans la mesure du possible, chacune de ces caractéristiques devrait être indépendante pour éviter l'interaction entre les gènes. Idéalement, il ne devrait pas y avoir de corrélation entre les valeurs des gènes. Dans ce cas hypothétique, la solution optimale peut être trouvée en sélectionnant indépendamment la meilleure valeur pour chaque gène. La figure (2.7), résume graphiquement la relation entre l'espace de solution d'un problème, la population de chromosomes avec laquelle l'algorithme génétique fonctionne et les fonctions de codification et de décodification[67] :

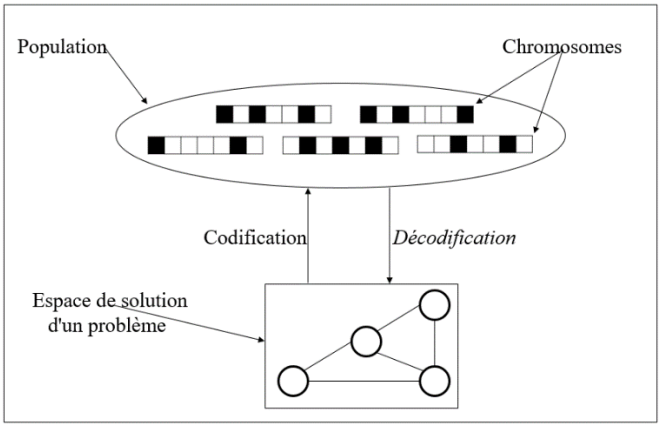

Figure 2.7 Codification de solutions dans un algorithme génétique

2.4.3 Population initiale

La construction de la population initiale est une opération très importante importance pour la performance d'AG, car elle contient des chromosomes dont la meilleure solution finale est faite. Généralement, les solutions initiales sont choisies au hasard, mais elles peuvent aussi être le résultat de certaines méthodes de construction[68].

Cependant, il faut être prudent d'utiliser une bonne population au début car ses solutions peuvent devenir trop tôt prédominantes dans cette population. Lorsque la population devient trop homogène, l'AG perd sa capacité de rechercher l'espace de solution jusqu'à ce que la population acquière lentement une certaine variation par la mutation. Récemment, les chercheurs ont fait de bons progrès avec AG parallèle[69], en utilisant plusieurs populations ou sous-populations qui évoluent indépendamment en utilisant différentes versions d'AG. Cependant, ce travail utilise une version séquentielle avec une seule population. La taille de la

population affecte la performance d'AG ainsi que le taux de convergence et le temps de fonctionnement[70]. Une population trop petite peut entraîner de mauvaises performances, car elle n'offre pas suffisamment de variété dans les solutions. Une population de grande taille fournit généralement de meilleures performances en évitant une convergence prématurée. La taille de la population est certainement parmi les paramètres qui nécessitent un réglage afin de trouver la valeur adaptée à chaque problème. Bien qu'une population constante soit utilisée ici, il est également possible d'utiliser une population dynamique, réduisant la taille de la population à mesure que le nombre d'itérations augmente. Il a été expérimenté que les améliorations les plus rapides de la population se produisent dans les premières itérations[71]. Ensuite, les changements deviennent de plus en plus petits et, en même temps, les individus les plus faibles deviennent de moins en moins significatifs. Dans chaque itération, un certain nombre de solutions parentes sont sélectionnées et un croisement et ou non d'autres opérateurs sont appliqués produisant des descendants. Le maintien de la population peut se faire de cette façon. La nouvelle population est sélectionnée parmi les solutions anciennes et les descendants. Afin de garder une taille de population constante, il est clair que certaines solutions dans la population précédente devront abandonner. Les algorithmes peuvent différer dans la façon dont une grande partie de la population est remplacée dans chaque itération.
La fonction population génère des solutions aléatoires. De nombreuses méthodes de sélection sont disponibles pour choisir les individus à reproduire et aussi pour survivre à la fin de chaque itération. Les mêmes parents peuvent être choisis plusieurs fois pour se reproduire. Les méthodes de sélection utilisent des valeurs de fitness associées à chaque solution pour comparer les solutions[67].

2.4.4 Fonction d'adaptation

La fonction de fitness quantifie l'aptitude de chaque chromosome comme solution au problème et détermine sa probabilité d'être sélectionné pour la phase de reproduction et de pouvoir transmettre une partie de son chromosome génétique à la génération suivante. Cette fonction est celle qui exerce une pression sur la population afin qu'elle évolue vers des chromosomes plus en forme, donc cette fonction est fondamentale pour le fonctionnement de l'algorithme. Comme le processus évolutif a tendance à retenir le chromosome génétique avec des valeurs de fitness élevées[72], un choix approprié de la fonction de fitness fournira une plus grande probabilité de conserver les caractéristiques des solutions quasi-optimales.

2.4.5 Mécanisme de sélection

Tout au long du processus suivi par un AG, il est souvent nécessaire de sélectionner des individus parmi ceux proposés par l'algorithme, soit pour se reproduire ou se recombiner, soit pour faire partie de la génération suivante proposée par un AG[73]. Ainsi, dans la majorité des AG, à partir d'une génération de départ, certains de leurs individus sont sélectionnés, en leur appliquant les opérateurs de recombinaison (croisement, mutation, etc.), en obtenant des nouveaux individus. Cette sélection peut être faite de différentes manières et selon différents critères, en fonction de la stratégie d'optimisation suivie par l'AG qui correspond le mieux à ses objectifs[74]. Normalement, l'opérateur de sélection est utilisé pour améliorer la qualité moyenne d'une population, car il attribue aux individus de la plus haute qualité une plus grande probabilité d'être sélectionné. Par conséquent, la perte de diversité génétique dans la population dépendra dans une large mesure du mécanisme de sélection utilisé. Normalement, les pires individus sont remplacés par les meilleurs, puis leur chromosome génétique disparaît après avoir effectué différentes phases de sélection. En d'autres occasions, tous les individus sont susceptibles d'être remplacés, bien qu'avec des probabilités différentes selon leur aptitude, donnant lieu à des populations plus diversifiées.

En résumé, il existe de nombreux mécanismes et algorithmes de sélection, ainsi que des moyens d'attribuer des probabilités de sélection à chacun des individus de la population.

Parmi ces mécanismes, il y a un processus de sélection proportionnelle appelé, la roulette, qui est une méthode fréquemment utilisée. Chaque individu est assigné par une valeur d'aptitude indiquant sa qualité. Dans la méthode de la roulette, la probabilité de choisir un individu est directement proportionnelle à sa valeur d'aptitude[75]. La figure (2.8) illustre la méthode d'une manière simple pour un problème ayant cinq individus dans une population. Considérant une épingle au sommet de la roue, on peut imaginer en tournant la roue qu'elle pointerait le plus fréquemment vers individuelle I_2 et que dans les occasions les plus rares pointeraient vers individuel I_1, I_4 ou I_5[66].

Par conséquent, celui qui présente la plus grande valeur de forme physique à plus de chances d'être sélectionné en tant que parent que celui ayant une faible valeur de forme physique.

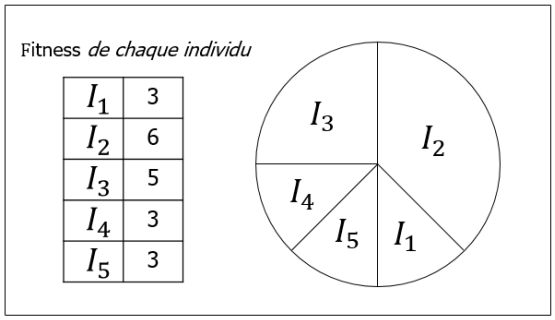

Figure 2.8 Illustration de la méthode de sélection par la roulette

Finalement, lorsque le mécanisme de sélection est déterminé, il faut définir les opérateurs de diversification de la population pour bien exploré l'espace de solutions.

2.4.6 Diversification de la population

La diversification de la population est une étape importante permettant l'exploration de l'espace de recherche. Cette diversification est garantie par les opérateurs de croisement et de mutation. Dans ce travail, un opérateur de croisement spécial qui respecte les principales propriétés de notre modèle est défini. Afin de garantir l'élément de codage spécifique de la population et de produire des descendants convenables[76], [77]. Le principal opérateur génétique est le croisement qui simule une reproduction entre deux chromosomes. Il travaille sur une paire de solutions et les recombine d'une certaine manière en générant un ou plusieurs descendants. Ces derniers partagent certaines des caractéristiques des parents et de cette manière les caractéristiques sont transmises aux générations futures. La fonctionnalité du croisement dépend de la représentation des données et la performance dépend de la façon dont il est ajusté au problème. De nombreux opérateurs de croisement différents ont été introduits dans la littérature. Afin d'aider à démontrer comment cela fonctionne, le simple croisement utilisé est illustré à la figure 2.9. L'illustration est faite avec la présentation de données binaires.

- **Croisement**

Le croisement simple commence avec deux solutions pères P_1 et P_2 et choisit une coupe aléatoire, qui est utilisée pour diviser les deux parents en deux parties. Il génère deux descendants D_1 et D_2 qui sont obtenus en regroupant une partie de père P_1 avec une partie de père P_2.

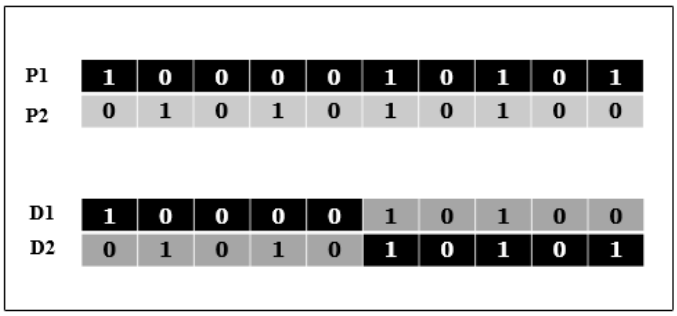

Figure 2.9 Operateur de croisement en un point

Figure (2.10) illustre l'opérateur de croisement. Le chromosome D_1 est généré à partir de la moitié gauche de P_1 et la moitié droite de P_2 et D_2 est faite à partir de la moitié droite de P_1 et de la moitié gauche de P_2.

L'opérateur de croisement recombine les gènes de deux individus existant dans la population. Cette recombinaison est appliquée dans une position spécifique qui s'appelle le point de croisement. Ce point est généré de façon aléatoire. Noter que, la sélection des deux parents pour produire deux enfants est réalisée d'une manière aléatoire.

- **Mutation**

Un autre opérateur est la mutation, qui est appliquée à une seule solution avec une certaine probabilité. Il fait de petits changements aléatoires dans la solution. Ces changements aléatoires ajouteront progressivement de nouvelles caractéristiques à la population, qui ne pourraient pas être fournies par le croisement. Il est important de mentionner que cet opérateur peut servir l'algorithme dans la recherche aléatoire. Une version très simple de l'opérateur est représentée à la figure (2.10).

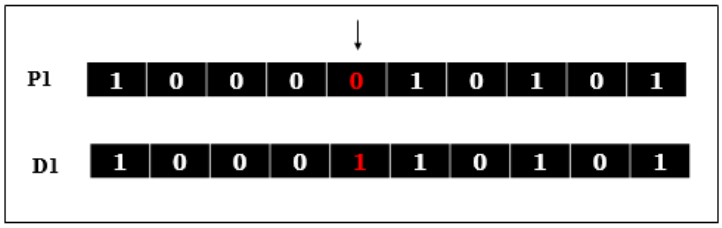

Figure 2.10 Operateur de mutation

Figure (2.10) illustre l'opérateur de mutation. Le bit de position 5 a été changé par 1 dans le descendant.

2.4.7 Convergence

Souvent l'évolution ressemble à un processus sans fin. On ne peut s'attendre à ce que les AG s'arrêtent spontanément à la solution optimale par chance. L'AG ne garantit même pas de trouver une telle solution. Par conséquent, l'évolution doit être arrêtée à un moment donné en fonction de certains critères préétablis. En fait, il existe plusieurs façons de décider quand arrêter l'algorithme. Le plus simple est d'arrêter l'évolution après un nombre fixe d'itérations. Une meilleure solution est de continuer les itérations tant que les améliorations restent appréciables. Enfin, on peut aussi attendre que la plupart ou la totalité des membres de la population soient similaires ou identiques, c'est-à-dire lorsque la population a convergé[71].
La convergence prématurée est une faute commune de l'AG, cela signifie généralement que la solution converge vers une solution inacceptable. Pour surmonter, ils peuvent être mis au point plusieurs techniques telles que l'augmentation du taux de mutation, ou mieux encore, en utilisant un taux de mutation adaptative. En règle générale, le taux de mutation doit être supérieur à la fin de la course pour éviter la perte-diversité de population.
En conclusion, seul un bon compromis entre la pression sélective, la taille de la population, le taux de mutation et d'autres paramètres, peut conduire à un algorithme approprié pour trouver de bonnes solutions dans une période de temps relativement courte.

2.5 Conclusion

Ce chapitre entreprend une étude exhaustive, d'une part de réseaux récurrent représenté par le réseau de Hopfield, et d'autre part les algorithmes évolutifs, en particulier les algorithmes génétiques. Nous avons présenté dans la première partie le réseau de Hopfield afin de déterminer sa capacité à résoudre des problèmes d'optimisation combinatoire. Nous avons présenté les deux modèles discrets et continus pour le système de Hopfield, ainsi que ses conditions de la convergence. L'analyse se concentrera sur le modèle continu, tandis que les aspects liés à la discrétisation seront laissés au chapitre de mise en œuvre de ce modèle. Afin d'appliquer les outils connus de la théorie des systèmes dynamiques, le réseau est formulé comme un système d'équation différentielle. La deuxième partie a été pour but d'étudier les différents opérateurs des algorithmes génétiques, représentation génétique, création de population, fonction d'adaptions, la sélection, la convergence et aussi sa flexibilité d'adaptation.

Chapitre 3 : Réseaux de contraintes

3.1 Introduction

La programmation par contraintes est un paradigme utilisé pour la description et résolution crédible et efficace de certains types de problèmes en général d'optimisation combinatoires. Ces problèmes apparaissent dans des domaines très divers, y compris l'intelligence artificielle, la recherche opérationnelle, les bases de données et les systèmes de récupération d'information, etc., avec des applications dans l'ordonnancement, la planification, le raisonnement temporel, la conception dans l'ingénierie, les problèmes d'emballage, la cryptographie, le diagnostic, la prise de décision, etc. Ces problèmes peuvent être modélisés comme des réseaux de contraintes (en anglais Constraint Network) et être résolus en utilisant des techniques de programmation par contraintes[78].

De nombreuses décisions que nous prenons en résolvant divers problèmes quotidiens sont soumises à des contraintes. Les décisions aussi courantes que la fixation d'un rendez-vous, la planification d'un voyage, l'achat d'une voiture ou la préparation d'une assiette peuvent dépendre de nombreux aspects interdépendants et même contradictoires, chacun étant soumis à un ensemble des contraintes qui doivent être respectées.

Les premiers travaux relatifs à la programmation par contraintes remontent aux années 60 et 70 dans le domaine de l'intelligence artificielle[79]. Au cours des dernières années, la programmation par contraintes a suscité une attention croissante en raison de son potentiel à résoudre des problèmes réels importants et complexes. Cependant, elle est considérée comme l'une des technologies les moins connues et les moins bien comprises. La programmation par contraintes est définie comme l'étude des systèmes de calcul basés sur des contraintes. L'idée de contraintes de planification est de résoudre des problèmes en déclarant des contraintes sur le domaine du problème et par conséquent de trouver des solutions aux instances du problème qui satisfont toutes les contraintes ou optimisent certains critères.

Durant les dernières années, un ensemble d'approches permettant de manipuler ces réseaux sont suffisamment étudiés[74]. Ces approches sont en général classifiées en sept classes de méthodes de résolution : méthodes complètes, méthodes de réduction, méthodes hybrides, méthodes incomplètes, Les méthodes d'ordonnancement de traitement des variables et valeurs, Les méthodes de décomposition structurelle et méthodes de programmation mathématique. Les méthodes complètes font une recherche arborescente en instanciant les variables une par une tout en satisfaisant les contraintes qui portent sur ces variables et en effectuant un retour en arrière en cas d'échec. Toutefois, les méthodes de réduction tentent de réduire le problème original dans le but de réduire la complexité de résolution via des méthodes de filtrage.

Les méthodes hybrides consistent à combiner les méthodes complètes et de réduction pour élaborer des nouvelles méthodes de résolution plus efficaces[80]. Les méthodes incomplètes font une réparation d'une solution partielle en parcourant de manière aléatoire l'espace de recherche tout en cherchant à améliorer la solution initiale. Pour la méthode d'ordre d'affectation de variable, l'attribution de variables est un aspect du problème de l'attribution d'algorithmes de recherche. Pour trouver une solution d'un système de contraintes et d'équations, l'algorithme de recherche doit affecter des valeurs aux variables afin de vérifier si l'objectif est atteint ou non (solution)[81]. Le problème d'affectation est une procédure séquentielle d'attribution de valeurs aux variables du problème. Par exemple, étant donné deux variables (x_1, x_2), nous pourrions d'abord attribuer la valeur à la variable x_1. Cependant, rien ne nous empêche d'attribuer la valeur à la variable x_2 en premier. Ce choix apparemment stérile peut avoir des effets très importants sur la complexité temporelle de l'algorithme de recherche. Surtout dans les algorithmes où il est nécessaire de trouver rapidement une solution. L'ordre des variables dans le processus d'affectation peut avoir un impact significatif sur l'efficacité et le temps de recherche. L'ordre des valeurs aussi peut fortement affecter l'efficacité, la complexité et le temps d'exécution de l'algorithme de recherche comme l'ordre d'affectation des variables[81].

Les méthodes de décomposition structurelle consistent à décomposer le problème initial sous forme moins complexe. En fin, des méthodes de programmation mathématique qui se basent sur la modélisation du problème en termes d'un problème de programmation quadratique et la résolution de ce modèle via des méthodes issues de la recherche opérationnelle[82].

Plusieurs extensions du formalisme CSP sont proposées dans la littérature[83][84]. Elles visent à élargir la portée de ce formalisme en introduisant de nouveaux concepts. Parmi ces

extensions, nous citons à titre d'exemple : Problèmes de satisfaction maximale de contraintes[85], dans des cas particuliers lorsqu'on modélise une application ou un problème réel sous forme d'un problème de satisfaction de contrainte, il arrive souvent qu'il n'existe aucune solution. Le problème est alors qualifié de sur-contrainte. Ce problème est connu dans le domaine de la recherche scientifique par le nom du problème de satisfaction maximale des contraintes (Max-CSP). Dans cette situation, on cherche souvent à trouver un compromis : on tolère une solution qui ne respecte pas toutes les contraintes. Le problème Max-CSP consiste à chercher une solution qui minimise le nombre de contraintes violées. Dans ce cadre, on cherche à minimiser le nombre de contraintes violées. Nous citons aussi, un autre type de CSP : Problèmes de satisfaction et d'optimisation sous contraintes[86]. Ce modèle incorpore une fonction objective et sert non seulement à satisfaire toutes les contraintes mais aussi d'optimiser cette fonction objective. Un autre type du problème de satisfaction de contraintes est donné par le problème de satisfaction de contraintes pondérées(WSCP) [84]. Il s'agit d'une extension du CSP qui permet de modéliser des préférences pour certains tuples de valeurs, ou encore pour représenter des coûts de violation de contraintes. Ces techniques offrent un cadre rigoureux et des algorithmes performants pour la recherche des solutions qui minimisent le coût des violations de contraintes. En général, Les CSP sont des problèmes importants et complexes, typiquement de complexité NP-complet[87]. Les étapes de base pour la résolution d'un problème CSP sont : modélisation et résolution en appliquant des techniques spécifiques à CSP.

Dans ce chapitre, nous présentons les concepts fondamentaux du CSP et ses algorithmes et techniques les plus pertinents du CSP, ainsi que divers exemples modélisés. Puis, nous étudions les extensions des problèmes de satisfaction de contraintes à savoir, les problèmes de satisfaction maximale de contraintes, les problèmes de satisfaction de contraintes pondérées, les problèmes d'optimisation de satisfaction de contraintes.

3.2 Définitions et concepts de bases

Le formalisme CSP est assez générique, il n'impose aucune limitation sur la nature et le nombre de contraintes, ni sur les types des domaines. La modélisation d'un problème sous forme d'un CSP nécessite l'identification des: variables, domaines de définitions des variables et les contraintes entre les variables. Le problème de satisfaction de contraintes a été introduit par Montanari[79]. Il est définit par un ensemble de variables et un ensemble de contraintes. Chaque variable peut prendre une valeur choisie dans le domaine de sa définition. Chaque

contrainte est définie sur un sous-ensemble de variables qui doivent satisfaire sa faisabilité. Pour résoudre le problème, on affecte une valeur à chaque variable de telle sorte que toutes les contraintes soient satisfaites. D'une manière très simple, ce problème est défini par un quadruplet (X, D, C, R) tel que :

- $X = \{x_1, \dots, x_n\}$ est un ensemble de n variables.
- $D = \{D(x_1), \dots, D(x_n)\}$ est un ensemble de définition pour chaque variable.
- $C = \{C_1, \dots, C_m\}$ est un ensemble de m contraintes. Chaque contrainte définie la relation existante entre certaines variables, restreignant les valeurs que peuvent prendre simultanément ces variables.
- $R = \{R_1, \dots, R_m\}$ est un ensemble de m relations, tel que chaque relation R_i est associée à une contrainte C_i. La relation R_i est définie sur $D(x_i) \times D(x_j)$, représentant les affectations possibles qui peuvent être prises par les variables tout en respectant la contrainte C_i.

Chaque variable possède ainsi un domaine initial qui contraint la valeur qu'on peut lui affecter. Un CSP est dit discret, si les valeurs de chaque domaine appartiennent à tout ensemble discret de valeurs, on parlera de variable discrète. La notion de domaine désigne l'ensemble de valeurs qui peuvent être affectées à une variable.

- **Remarque**

Lorsque le domaine d'une variable est réduit à une valeur, on dit que la variable est instanciée. Une fois qu'on a défini pour chaque variable son domaine initial, on peut définir des contraintes reliant les variables du problème et restreignant les combinaisons de valeurs possibles. Dans ce cadre, la définition de chaque contrainte peut se faire par la donnée de l'ensemble des tuples autorisés (ou interdits), ou par la fonction caractéristique de cet ensemble. Par la suite, une série des définitions est obligatoire dans le but de bien décrire les propriétés fondamentales et complémentaires d'un problème de satisfaction de contraintes.

- **Définition – Arité d'une contrainte**

L'arité d'une contrainte est le nombre de variables impliquées par une contrainte, autrement dit, c'est le nombre de variables qui entre en relation pour définir une contrainte. Ces variables sont regroupées dans un ensemble noté $vars(C)$.

Via l'arité, on peut définir une classification des contraintes. Donc, une contrainte s'appliquant seulement à une variable sera dite unaire tandis que, pour une contrainte

s'appliquant à deux variables, on parlera d'une contrainte binaire et ainsi de suite, si l'arité d'une contrainte est égale à n, alors on parlera d'une contrainte n-aire. De plus, en fonction des domaines de valeur des variables, on distingue différents types de contraintes : contraintes numériques, contraintes booléennes, contraintes symboliques, etc.

- **Définition – CSP binaire**

Un CSP binaire est un CSP dont toutes les contraintes $C_{ij} \in C$ ont une arité inférieure ou égale à deux, c'est à dire, chaque contrainte est définie exactement à base d'une variable ou deux variables $x_i \; et \; x_j$.

Sachant que si l'arité d'une contrainte est inférieure strictement à deux, dans ce cas, cette contrainte importe des restrictions dans le domaine de définition de la variable correspondante. Dans le cas où l'arité d'une contrainte dans le problème est strictement supérieure à deux, le problème de satisfaction de contraintes est connu comme un CSP n-aire. Rappelant qu'il est possible de convertir un CSP avec des contraintes n-aire en un CSP binaire équivalent [88]. C'est pour cela que la plupart des recherches pour la résolution de CSP s'intéresse au problème de satisfaction de contraintes binaire[89].

- **Définition – Instanciation**

Une instanciation est une application qui associe à chaque variable x_i une valeur de son domaine $D(x_i)$.

- **Définition – Instanciation partielle**

Etant donné un CSP : une instanciation partielle est une instanciation d'un sous-ensemble des variables, c'est-à-dire une sous-séquence d'instanciation.

- **Définition – Instanciation totale**

Une instanciation totale est une instanciation de toutes les variables $x_i \in X$.

- **Définition – Satisfaction d'une contrainte**

Etant donné un CSP : la satisfaction d'une contrainte C_{ij} est une instanciation partielle de deux variables $x_i \; et \;\; x_j$ par les valeurs qui sont respectivement $v_r \in D(x_i) \; et \; v_s \in D(x_j)$, tel que le couple $(v_r, v_s) \in R_{ij}$.

- **Remarque**

Si la contrainte C_{ij} n'est pas satisfaite alors on dit qu'elle est violée. C'est-à-dire, que l'instanciation partielle des deux variables $x_i \; et \; x_j$ par les valeurs respectivement $v_r \in D(x_i) \; et \; v_s \in D(x_j)$ n'appartient pas à la relation $R_{ij} : (v_r, v_s) \notin R_{ij}$.

- **Définition - Espace de recherche**

L'espace de recherche d'un CSP est l'ensemble des affectations possibles qu'on peut noter par E_{CSP} avec $E_{CSP} = D(x_1) \times D(x_2) \times ... \times D(x_n)$. L'espace de recherche est égal au produit cartésien de l'ensemble des domaines des variables et une instanciation totale (IT) sera assimilée à un élément de cet ensemble E_{CSP}.

La résolution d'un CSP consiste soit de trouver une solution à condition que le problème est consistant, soit de prouver qu'il n'existe pas de solution si le problème est inconsistant. Donc une solution d'un CSP est définie par :

- **Définition – Solution d'un CSP**

La solution d'un CSP est une satisfaction de toutes les contraintes $C_{ij} \in C$. Par la suite, toute approche de résolution d'un CSP sera nommée par le mot solveur. La structure de ce solveur peut être définie, en générale, de la manière suivante :

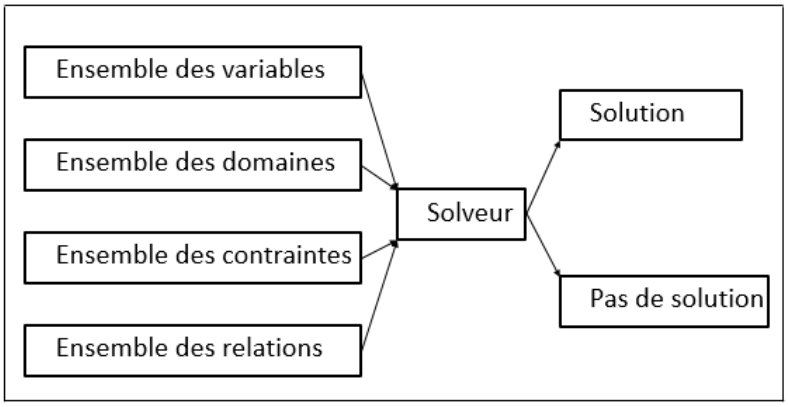

Figure 3.1 Structure d'un solveur de CSP

Un CSP binaire peut être représenté par un graphe de contraintes dont la définition de ce dernier est la suivante.

- **Définition – Graphe de contraintes**

A chaque CSP binaire, on peut associer un graphe des contraintes. Ce graphe est obtenu en représentant chaque variable du réseau par un sommet et chaque contrainte binaire qui porte sur les variables par un arc entre les sommets.

- **Remarque**

Dans le cas des problèmes CSP avec des contraintes n-aire, on peut utiliser la représentation par hypergraphe, en remplaçant les arcs par des hyper-arcs.

Exemple 1 : Soit le problème de satisfaction de contraintes défini par un quadruplet (X, D, C, R) tel que :

- $X = \{x_1, x_2, x_3\}$ est un ensemble de 3 variables.
- $D = \{D(x_1), D(x_2), D(x_3)\}$ tel que chaque domaine $D(x_i)$ est un ensemble de valeurs possibles pour la chaque variable x_i. Les domaines de définition contiennent les valeurs suivantes : $D(x_1) = D(x_2) = D(x_3) = \{1,2,3\}$.
- $C = \{C_{12}, C_{13}, C_{23}\}$ est un ensemble de 3 contraintes. Chaque contrainte est une relation entre deux variables, restreignant les valeurs que peuvent prendre simultanément ces variables tels que :

C_{12} est une contrainte définissant la relation de différence entre les variables $x_1 et\ x_2$ $(x_1 \neq x_2)$. Les couples satisfaisant la contrainte C_{12} sont définis dans la relation R_{12} suivante :

$$R_{12} = \{(1,2), (1,3), (2,1), (2,3), (3,1), (3,2)\}$$

C_{13} est une contrainte définissant une relation de supériorité stricte entre les variables $x_1 et\ x_3$ $(x_1 < x_3)$. La relation R_{13} qui exprime explicitement cette contrainte est :

$$R_{13} = \{(1,2), (1,3), (2,3)\}$$

C_{23} est une contrainte définissant une relation d'égalité entre les variables $x_2\ et\ x_3$ $(x_2 = x_3)$. La relation R_{23} qui exprime explicitement cette contrainte est :

$$R_{23} = \{(1,1), (2,2), (3,3)\}$$

Dans l'exemple ci-dessus, une contrainte C_{ij} entre deux variables $x_i\ et\ x_j$ est définie par l'ensemble des couples (v_r, v_s) qui la satisfont, le premier (resp. deuxième) élément du couple étant une valeur du domaine de $x_i\ (resp.\ x_j)$. Ce problème utilise des variables à domaine numérique et des contraintes définissant explicitement les combinaisons de valeurs autorisées.

Chacune de ces affectations $x_1 = 3,\ x_2 = 1\ et\ x_3 = 2$ est une instanciation. De plus, les trois instanciations est une instanciation totale.

L'instanciation partielle $x_1 = 2\ et\ x_2 = 1$ conduit à une satisfaction de la contrainte C_{12} car le couple $(2,1) \in R_{12}$.

L'instanciation totale $x_1 = 1,\ x_2 = 3\ et\ x_3 = 3$ est une solution du problème CSP car les couples $(1,3) \in R_{12},\ (1,3) \in R_{13}\ et\ (3,3) \in R_{23}$.

Par contre, L'instanciation totale $x_1 = 2,\ x_2 = 1\ et\ x_3 = 1$ ne représente pas une solution du problème CSP car les couples $(2,1) \in R_{12},\ (2,1) \notin R_{13}\ et\ (1,1) \in R_{23}$.

Dans cette section, nous avons bien introduit le formalisme CSP avec des définitions, des remarques et un exemple. Ce formalisme est utilisé pour modéliser de nombreux problèmes issus du monde réel. Dans la section qui suit, quelques problèmes académiques sont modélisés en terme d'un problème de satisfaction de contraintes.

3.3 Exemples de problèmes modélisés comme des problèmes CSP

La première étape d'utilisation des CSP, comme modèles génériques, est la modélisation. C'est-à-dire la représentation en terme de variables, de domaines et de contraintes. Comme pour le langage naturel, la modélisation d'un problème peut se faire de différentes manières. En ce qui concerne la modélisation d'un problème CSP, il y a deux aspects fondamentaux :

- Le pouvoir expressif des contraintes, c'est-à-dire la capacité à modéliser les contraintes existant réellement dans le problème réel.
- L'efficacité de la représentation avec laquelle le problème sera résolu avec plus ou moins d'efficacité.

De nombreux problèmes académiques et réels sont modélisés sous forme de CSP. Dans cette section, nous présentons quelques problèmes typiques et leurs modèles de CSP équivalents.

3.3.1 Placement de reines

Ce problème consiste à placer n reines sur un échiquier de dimension $n \times n$, de sorte qu'aucune reine ne soit menacée. De cette façon, il ne peut pas être deux reines dans la même ligne, la même colonne ou la même diagonale[90], [91]. Si nous associons chaque colonne à une variable et sa valeur représente la ligne où une reine est placée, Le problème peut être formulé comme[92]:

- Variables $X = \{x_i\}, i = 1, \dots, n$

- Domaines $D(x_i) = \{1, \dots, N\}, i = 1, \dots, N$
- Contraintes C: $(\forall x_i, \forall x_j \; et \; i \neq j)$
  - C_1: $x_i \neq x_j$ exprimant que deux reines ne doivent pas être sur la même ligne.
  - C_2: $x_i - x_j \neq i - j$ exprimant que deux reines ne doivent pas être sur la même diagonale sud-est.
  - C_3: $x_j - x_i \neq i - j$ exprimant que deux reines ne doivent pas être sur la même diagonale sud-ouest.

Résultant dans le cas particulier de $N = 4$:

- Variables $X = \{x_1, x_2, x_3, x_4\}$
- Contraintes C_1: $x_1 \neq x_2, \; x_1 \neq x_3, x_1 \neq x_4, \; x_2 \neq x_3, \; x_2 \neq x_4 \; et \; x_3 \neq x_4$

Les contraintes $C_2 \; et \; C_3$ représentées par :

$$|x_1 - x_2| \neq 1 \, , |x_1 - x_3| \neq 2 \, , |x_1 - x_4| \neq 3$$

$$|x_2 - x_3| \neq 1 \, , |x_2 - x_4| \neq 2 \, , |x_3 - x_4| \neq 1$$

La relation $R = \{R_{12}, R_{13}, R_{14}, R_{23}, R_{24}, R_{34}\}$ définie les couples possibles à instancier :

$$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$$
$$R_{13} = \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\}$$
$$R_{14} = \{(1,2), (1,3), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4), (4,2), (4,3)\}$$
$$R_{23} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$$
$$R_{24} = \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\}$$
$$R_{34} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$$

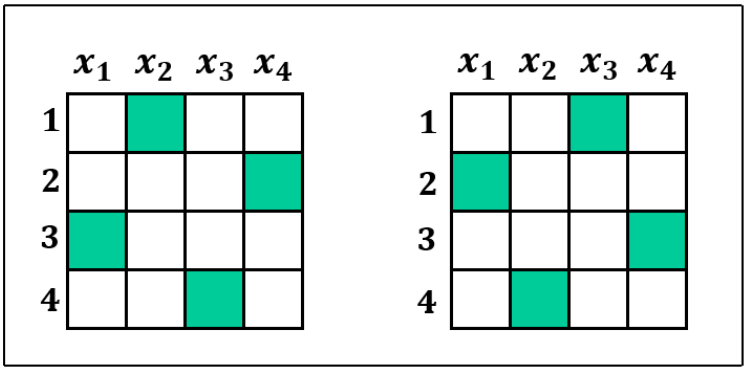

Figure 3.2 Solution pour le problème de 4 reines sur un échiquier

Par conséquent, ce problème de reines est représenté sous forme d'un problème CSP. Donc, toute solution de ce dernier est une solution du problème original.
Il y a deux solutions $\{x_1 = 2,\ x_2 = 4,\ x_3 = 1\ et\ \ x_4 = 3\}\ et\ \{x_1 = 3,\ x_2 = 1,\ x_3 = 4\ et\ \ x_4 = 2\}$ pour le problème des 4 reines. La figure (3.2) montre la représentation graphique pour ces deux solutions.

3.3.2 Coloriage de graphes

Ce problème commence à partir d'un ensemble de couleurs possibles pour colorer chaque région d'une carte, de sorte que les régions adjacentes ont des couleurs différentes. Dans la formulation du CSP, nous définissons une variable pour chaque région de la carte, et le domaine de chaque variable est l'ensemble des couleurs disponibles[93], [94]. Pour chaque paire de régions contiguës, il existe une contrainte sur les variables correspondantes qui ne permet pas l'attribution de valeurs identiques aux variables. La figure (3.3), montre un exemple de problème du coloriage de carte de notre région Fès-Meknès. La carte comporte six régions qui doivent être coloriées en rouge, bleu, vert, blanche ou noir. Il faut rappeler que nous avons choisi seulement cinq couleurs parce que le nombre maximum des régions adjacentes est égal à cinq. La reformulation en termes d'un CSP consiste à introduire une variable pour chacune des régions. Le domaine de chaque variable correspond à l'ensemble des couleurs. Pour chaque paire de régions adjacentes, une contrainte binaire est imposée entre les variables correspondantes. Cette contrainte interdit toute affectation de la même couleur à deux variables correspondantes.

Figure 3.3 Problème de coloriage de carte de la région Fes-Meknes

Le problème de satisfaction de contrainte qui traduise ce problème de coloriage de la carte peut être défini de la façon suivante :

- $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
- $D(x_i) = \{bleu, rouge, vert, blanche\ et\ noir\}\ et\ i = 1,2, \dots, 6$
- C est l'ensemble de contraintes défini par :

  $x_1 \neq x_2\ , x_1 \neq x_5\ , x_2 \neq x_5\ , x_2 \neq x_3\ , x_2 \neq x_4\ , x_3 \neq x_4\ , x_5 \neq x_6\ , x_5 \neq x_4\ et\ x_6 \neq x_4$
- $R = \{R_{12}, R_{15}, R_{25}, R_{23}, R_{24}, R_{34}, R_{56}, R_{54}, R_{64}\}$ est l'ensemble de relations contenant les couples possibles. Donc une solution à ce problème CSP, est une solution au problème de coloriage de la carte.

3.3.3 Problème du nombre de Langford

Le problème du nombre de Longford a été publié en 1958 par le mathématicien écossais C.D. Langford qui a observé des alignements de cubes de couleurs rouge, bleue et jaune réalisés par son fils[95]. En utilisant des nombres à la place des couleurs, ce problème consiste à placer n ensembles de chiffres comportant chacun les chiffres de 1 à k tout en respectant un ordre de sorte que les deux 1 soient séparés par un chiffre, les deux 2 par deux chiffre, etc.

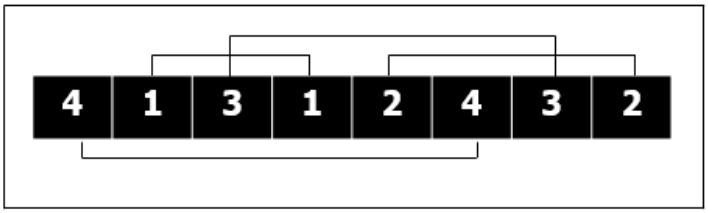

Figure 3.4 Solution du nombre de Langford (2,4)

La figure 3.4 représente une solution du problème avec deux ensembles de quatre chiffres. Une formalisation en CSP comprend deux ensembles avec quatre chiffres. Huit variables de x_1 à x_8 sont nécessaires pour cette modélisation. Le domaine de chaque variable correspond aux places que peut prendre la variable dans la séquence. Les contraintes se définissent alors comme suit :

- $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$
- $D(x_i) = \{1,2,3,4\}\ et\ i = 1,2, \dots,8$

L'ensemble de contraintes et relations est défini par :

$$R_{ij} \Leftrightarrow \{x_i \neq x_j\ , x_i + (i+1) \neq x_j + (j+1)\ ,\ x_i + (i+1) \neq x_j\ et\ x_i \neq x_j + (j+1)\}$$

Ces exemples d'application font partie d'une grande classe de problèmes du monde réel qui peuvent être vues comme des problèmes de satisfaction de contraintes. Par conséquent, les chercheurs ont beaucoup investis durant des années et des années de recherche pour résoudre efficacement ce type de problèmes.

3.4 Approches de résolution des problèmes CSP

Il existe des algorithmes spécifiques pour trouver des solutions aux problèmes de satisfaction de contraintes. Ces algorithmes suivent différentes méthodologies pour chercher la solution optimale ou approchée. La solution est obtenue par une instanciation totale en respectant toutes les contraintes. L'instanciation de toutes les variables permet de générer un espace de recherche qui est représenté sous forme d'un arbre. Et à base de cet arbre que les algorithmes de recherche effectuent leurs trajets pour la recherche d'une solution[96].

Nous distinguons deux principaux types d'algorithmes permettent la recherche d'une solution. Ceux-ci sont les algorithmes complets et incomplets. La principale différence entre les deux réside dans le domaine de la recherche de la solution. Les algorithmes incomplets sont aussi appelés locaux parce qu'ils ne font que chercher dans un sous espace de l'espace de recherche. Donc ils ne garantissent pas une solution optimale, solution approchée, mais ils ont l'avantage de réduire le coût de calcul en parcourant un espace réduit et cette technique de recherche permet de trouver la solution dans un temps raisonnable. Le fonctionnement de cette technique consiste à trouver la solution à partir d'une solution initiale, qui itère plusieurs fois en s'adressant à chaque fois à une solution différente, essayant d'améliorer la valeur de la fonction objectif. Le deuxième groupe comprend les algorithmes complets qui parcourent tout l'espace à la recherche d'une solution en assignant des valeurs aux variables dans un arbre. Le nœud racine de l'arbre représente le problème sans variables attribué. Dans le but de résoudre CSP, un grand nombre d'algorithmes de recherche complets ont été développés qui travaillent sur l'ensemble de l'espace de recherche dont certains sont plus efficaces que d'autres. Pour simplifier l'explication des algorithmes de recherche, nous utiliserons le même exemple que précédemment utilisé dans CSP, de cette façon il est plus facile de comprendre le fonctionnement des différents algorithmes qui recherchent des solutions possibles.

3.4.1 Recherche systématique

La recherche systématique consiste à parcourir l'espace de recherche jusqu'à trouver une solution ou prouver qu'il n'y a pas de solution.

Les combinaisons possibles d'attribution de valeurs à des variables dans un CSP [89] donnent lieu à un espace de recherche qui peut être représenté comme un arbre de recherche ou un graphe. Chaque nœud de l'arbre de recherche représente une attribution partielle de valeurs à un ensemble de variables. Le nœud racine de l'arbre de recherche représente le cas dans lequel aucune variable n'est instanciée. Il existe plusieurs algorithmes de recherche systématique destinés à cette mission[97].

a) Génère et teste

Génère et teste est la technique de recherche la plus ancienne utilisée car elle instancie chacune des valeurs possibles sur les variables et parcourt systématiquement tout l'arbre de recherche. Son nom est dérivé des deux actions principales qu'elle réalise. La première action consiste à instancier chaque variable par une valeur de son domaine, cette instanciation commence de nœud racine de l'arbre jusqu'à les feuilles. La seconde action consiste à tester ou vérifier la conformité des contraintes. Cette méthode présente l'inconvénient d'être très onéreuse et lente lors de la recherche d'une solution. Cet inconvénient peut être traduit par le nombre des instanciations générées et qui ne respectent pas les contraintes, cela peut conduire une perte de temps et de coût.

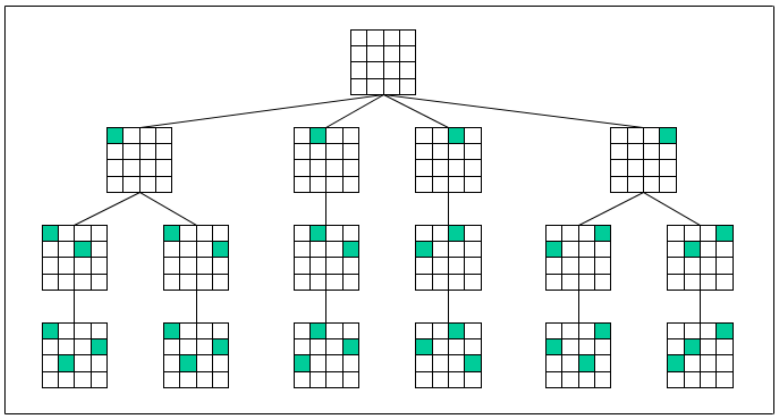

Figure 3.5 Génère et teste pour le problème des 4 reines

En appliquant l'algorithme génère et test (GT) dans le problème des n-reines un grand nombre d'instances inutiles est conçu. L'algorithme GT parcourt chaque branche de l'arbre pour générer une solution sans une vérification des contraintes ce qui montre l'inefficacité de cette méthode. La figure (3.5) représente le problème de 4 reines avec l'ensemble des solutions possibles.

b) Algorithme de backtraking

Un autre algorithme de recherche de CSP est bien connu sous le nom bakctracking (BT) [21] [22]. Le fonctionnement de BT est similaire à l'algorithme de génère et teste mais avec la grande différence qu'il parvient à éliminer l'inefficacité de la GT.

Une recherche en profondeur est poursuivie pour chaque nœud sachant que, pour chaque valeur affectée à une variable (x_i) avec laquelle la recherche est en cours, une vérification de la contrainte actuelle pour une solution partielle est réalisée (sans instancier toutes les variables). Si la valeur affectée à la variable x_i produit une incohérence, en tenant compte des variables affectées précédemment, l'algorithme de backtraking (BT) choisi une nouvelle valeur du domaine de la même variable x_i parce que l'instanciation partielle ne fait partie d'aucune solution. Cet algorithme permet d'éviter l'itinéraire qui ne mène pas à une solution. Ce processus est fait jusqu'à l'épuisement de chacune des possibilités du domaine $D(x_i)$ de la variable courante.

Si les possibilités sont épuisées pour une variable alors BT revient en arrière et change la valeur de la variable précédente. De cette manière, le fonctionnement de l'algorithme se poursuit jusqu'à ce qu'une solution soit trouvée, ce qui est réalisé lorsque chacune des affectations courantes faites en conjonction avec les affectations passées, sont systématiquement en accord avec les contraintes du problème. Si toutes les possibilités sont épuisées pour toutes les variables, il est démontré qu'il n'y a pas de solution au problème. Le processus d'exploration d'arbre pour l'algorithme BT se résume en deux phases [98]:

– **Phase d'instanciation** : permettant d'affecter une valeur du domaine à la variable courante dans le but d'étendre la sous-séquence,

– **Phase de retour en arrière** : consiste à revenir à la dernière variable affectée.

Le graphe de résolution est illustré dans la Figure (3.6), montrant l'étape de choix de la valeur à affecter à la prochaine variable appliqué au problème des 4-reines.

Dans l'exemple de problème des 4-reines, l'algorithme place la première reine en haut à gauche de l'échiquier, ensuite la deuxième reine trouve de place dans une position de la ligne suivante et ainsi de suite. Au cas d'inconsistance, l'algorithme entraine un retour à la position précédente.

Cet algorithme donne une bonne réponse à tous les coups, mais conduit à tester un grand nombre de solutions potentielles, ce qui ramène à une croissance exponentielle et par conséquent, rendant souvent les traitements très lourds. Cet algorithme teste toutes les valeurs

possibles avant de trouver une solution. Compte tenu de la taille et de la dureté des problèmes gérés, ses dimensions deviennent rapidement non-maitrisables. Si jamais la solution se trouve dans les dernières possibilités ou si jamais il n'y a pas de solutions, le temps de calcul devient inacceptable.

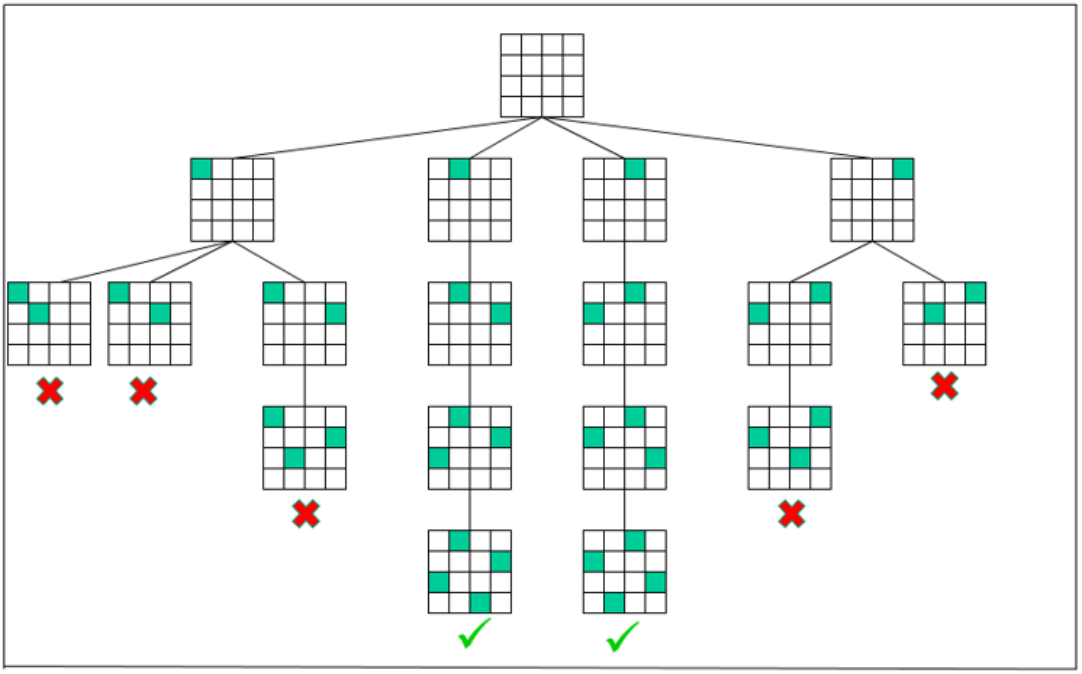

Figure 3.6 Arbre de recherche de backtraking

Il faut noter que de nombreux algorithmes sont élaborés et développés à base de l'algorithme BT pour résoudre le problème de CSP. En autre, certains chercheurs ont travaillé durement sur la phase de retour en arrière pour augmenter l'efficacité de résolution[98]. D'autres ont élaboré des procédures de prévision de violation d'une contrainte par l'ajout de méthode de filtrage de domaines visant la suppression de valeurs rendues impossibles par l'état courant d'une résolution[99]–[101].

c) Algorithme de back-Jumping

Lorsque l'affectation d'une variable est échouée, le retour en arrière (backtracking) n'effectue qu'un simple retour en arrière en mettant en cause la variable instanciée et essaie alors une nouvelle valeur pour la variable courante. Lorsque toutes les valeurs ont été utilisées, il revient en arrière sur la variable précédente. Cependant, rien ne garantit que cette variable soit en cause dans les différents échecs rencontrés lors de l'affectation. Pour pallier à ce problème

du Back-Tracking [99]–[101], plusieurs méthodes dotées de retour arrière non chronologique ont vu le jour. Dans ce contexte, une analyse permettra un saut vers un niveau plus élevé s'il s'avère que la cause d'échec est en amont dans l'arbre de recherche. C'est la notion de saut en arrière ou back-jumping.

Les algorithmes de Back-Jumping (BJ) ont été développés dans le but d'accélérer le processus de retour en arrière. Cet algorithme conserve, comme le BT standard, une étape de choix de la prochaine variable à instancier et une étape de retour en arrière lors de situations d'inconsistance. Cependant, plutôt que d'effectuer un retour en arrière vers la $(i-1)^{éme}$ variable instanciée, il utilise un principe de saut vers une variable plus loin dans la sous-séquence d'instanciation. Le BJ a comme but d'identifier la variable ayant causé la violation de la contrainte. Pour cela une analyse des raisons de la situation de violation et la recherche de la variable ayant eu le plus d'impact sur celle-ci est faite. Plusieurs algorithmes utilisant le principe de BJ ont été développés[102].

d) Algorithmes avec mémorisation

Les algorithmes avec retour en arrière non chronologique permettent d'éviter certaines redondances dans l'arbre de recherche. Toutefois, il reste des cas là où il tombe dans certaines redondances. Pour pallier à ce point négatif et pour ne pas visiter plusieurs fois les mêmes sous arbres, une solution consiste à mémoriser des informations portant sur le parcours déjà entamé. Ces informations sont généralement mémorisées sous la forme de contraintes induites, qui sont souvent désignées sous le terme de no-good. Un no-good correspond à une affectation qui ne peut être étendue en une solution. Leur mémorisation permet donc d'interdire certaines affectations et ainsi d'éviter de parcourir plusieurs fois les mêmes sous arbres.

Finalement, on peut conclure que toutes ces méthodes de la recherche systématique se basent sur la notion d'exploration d'arbre. Les améliorations dans ce cadre sont basées sur l'action de retour. Ces méthodes sont connues par leurs inefficacités devant des problèmes de grande taille malgré les améliorations proposées dans ce contexte. Donc une combinaison avec des techniques de filtrage sera bénéfique pour résoudre efficacement ce type de problèmes.

3.4.2 Techniques de consistance

Les techniques de consistance ont d'abord été introduites dans l'intelligence artificielle pour améliorer l'efficacité des programmes de reconnaissance d'images[1]. Diverses techniques de

consistance locale ont été proposées dans la littérature pour améliorer l'efficacité des algorithmes de recherche. Ces techniques sont utilisées comme étapes de prétraitement où les inconsistances locales sont détectées et éliminées, avant de lancer la recherche ou pendant le processus de recherche lui-même, afin de réduire les nœuds à instancier dans l'arbre de recherche.

Nous pouvons différencier différents niveaux de consistance locale comme une consistance de nœud, une consistance d'arc ou une consistance de chemin. Les algorithmes qui atteignent de tels niveaux de consistance éliminent les instanciations de valeurs dans les domaines des variables incompatibles, c'est-à-dire qu'elles éliminent les nœuds de l'arbre de recherche, qui ne peuvent participer à aucune solution.

De tels algorithmes sont appelés algorithmes de propagation de contraintes, algorithmes de filtrage ou algorithmes de consistance. Bien que nous appliquions les algorithmes de consistance, nous ne garantissons pas que toutes les paires variable-valeur restantes fassent partie d'une solution, la pratique a montré qu'elles peuvent être très utiles en tant qu'étape de prétraitement, pour réduire la complexité des CSP et aussi pendant la recherche, pour élaguer l'espace de recherche. Dans cette sous-section, nous allons passer en revue quelques travaux antérieurs sur les algorithmes de consistance.

a) Consistance de nœud

Le niveau de base de consistance est la consistance de nœud où les valeurs inconsistantes sont éliminées avec les contraintes unaires auxquelles la variable participe.

Une variable est consistante avec le nœud si et seulement si toutes les valeurs de domaine satisfont la contrainte unaire à laquelle la variable participe[103].

- **Exemple**

Soit la variable x_1 avec un domaine discret défini par : $D(x_1) = \{-2, -1,0,2,4,5\}$ et une contrainte unaire liée associée à cette variable est donné par $R_1: x_1 > 3$. La consistance du nœud va éliminer les valeurs $\{-2, -1, 0, 2\}$ du domaine de x_1. La figure 3.7 (a) montre le problème original et la figure 3.7 (b) montre le résultat de l'application de consistance de nœud à la variable x_1.

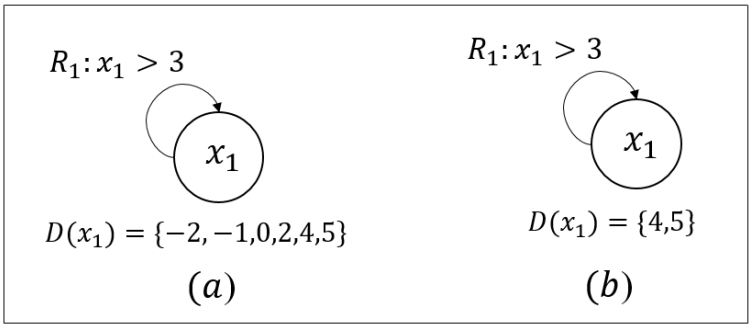

Figure 3.7 Consitance de nœud : (a) domaine d'origine et (b) domaine consitant de nœud

b) Consistance d'arc

Un CSP est consistant d'arc entre le couple de variables (x_i, x_j) si pour chaque valeur $a\ dans\ D(x_i)$ il y a au moins une valeur $b\ dans\ D(x_j)$ telle que l'affectation $(x_i, a)\ et\ (x_j, b)$ vérifie la contrainte entre x_i et x_j. Toute valeur dans le domaine $D(x_i)$ de la variable x_i qui n'est pas consistante avec l'arc doit être éliminée de $D(x_i)$ car elle ne peut faire partie d'aucune solution. Le domaine d'une variable est consistant avec l'arc si toutes ses valeurs sont consistantes avec l'arc. Ainsi, un problème est un arc-consistant si et seulement si tous ses arcs sont consistants[104].

Dans l'exemple de la figure (3.8), nous pouvons voir que le problème est un arc consistant, puisque pour chaque valeur $a \in [1,4]$ il y a au moins une valeur $b \in [5,8]$ pour que la contrainte soit satisfaite $x_i < x_j$. Cependant, si la contrainte est $x_i = x_j$ alors elle n'est pas consistante avec l'arc.

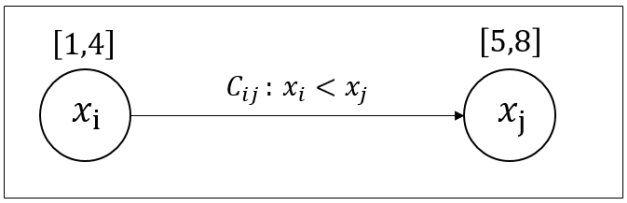

Figure 3.8 Consistance d'arc

c) Consistance du chemin

La consistance du chemin est un niveau de consistance locale plus élevé que la consistance d'arc[105]. La consistance du chemin pour chaque paire de valeurs a et b de deux variables $x_i\ et\ x_j$ si pour chaque instanciation $< x_i, a > et\ < x_j, b >$ vérifie la contrainte entre $x_i\ et\ x_j$ et pour chaque valeur d'une variable tout au long du chemin entre $x_i\ et\ x_j$ toutes les contraintes sur le chemin sont satisfaites. Lorsqu'un problème satisfait la

consistance du chemin alors chaque couple de variables dans le chemin est un consistant d'arc La figure (3.9) représente la consistance du chemin entre deux variables x_i et x_j.

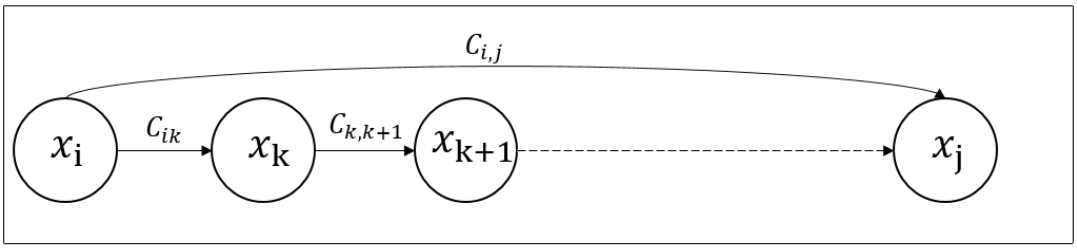

Figure 3.9 Consistance de chemin

La consistance d'arc est classifiée parmi les techniques les plus étudiées et les plus utilisées pour effectuer des filtrages lors de la résolution de CSP. Cette technique est applicable sur tous les types de CSP discrets. Ce qui est reflété par l'existence de plusieurs algorithmes de conception simple et capables d'établir la consistance d'arc d'une manière optimale[106]. La consistance d'arc est définie sur les problèmes de satisfaction de contraintes binaires avec la possibilité de généraliser celle-ci pour filtrer des problèmes CSP non binaires.

3.4.3 Techniques hybrides

Les techniques hybrides se basent sur une combinaison des techniques de la recherche systématique et les techniques de consistance dans le but de prévoir la violation d'une contrainte avant l'instanciation. Dans ce cadre, plusieurs algorithmes sont élaborés à base de cette notion de vérification. L'algorithme de Forward-Checking et Look-Ahead sont les plus connus dans ce domaine[107].

a) Algorithme de forward checking

L'algorithme de saut en arrière (BackJumping) permet d'instancier une variable à la fois et appliquer le schéma de retour en arrière lorsque la variable courante a un domaine vide. Alors les décisions sont faites et on subit les impacts de ces décisions par la suite. Toutefois, le forward-Checking (FC) a pour but d'éviter à l'avance des instanciations inconsistantes en appliquant le critère de la consistance d'arc pendant la recherche. Cet algorithme essaye de prévoir l'impact d'une instanciation avant même que la décision soit prise. Avant d'affecter une valeur v_r à une variable x_i, Il vérifiera l'impact sur les domaines pour toutes les autres variables non encore traitées. Si la valeur que l'on désire affecter à la variable conduit à un domaine vide d'une autre variable, donc il est naturel de changer cette valeur

d'instanciation[107]. L'algorithme FC utilise une méthode nommée Forward-Checking qui permet de vérifier si l'instanciation courante est possible et, dans l'affirmative, passera à l'instanciation suivante[108]. Dans le cas contraire, il appellera une autre méthode nommée restauré qui restaurera le domaine de la variable x_i puisqu'on n'est pas parvenu à l'instancier. Finalement, Le FC effectuera un filtrage des valeurs de chacune des variables qui sont inconsistantes avec l'instanciation désirée et supprimera ces valeurs. Il y'aura donc une diminution des domaines des variables et permettra de détecter de nombreuses inconsistances plus tôt qu'avec un algorithme de Back-Jumping. De plus, la complexité en temps de FC dans le pire des cas est identique à celle de BT[109]. Par contre, en pratique, FC obtient généralement de meilleurs résultats que BT.

La figure (3.10), montre l'arbre de recherche du Forward-Checking. A chaque instanciation d'une variable, les conséquences à court terme sont propagées sur les autres domaines. Une reine est placée d'abord en haut à droite de l'échiquier, les contraintes impliquant cette reine sont alors utilisées et suppriment la possibilité de placer la seconde dans la première ligne et la première colonne et la diagonale qui passe par la première reine, etc.

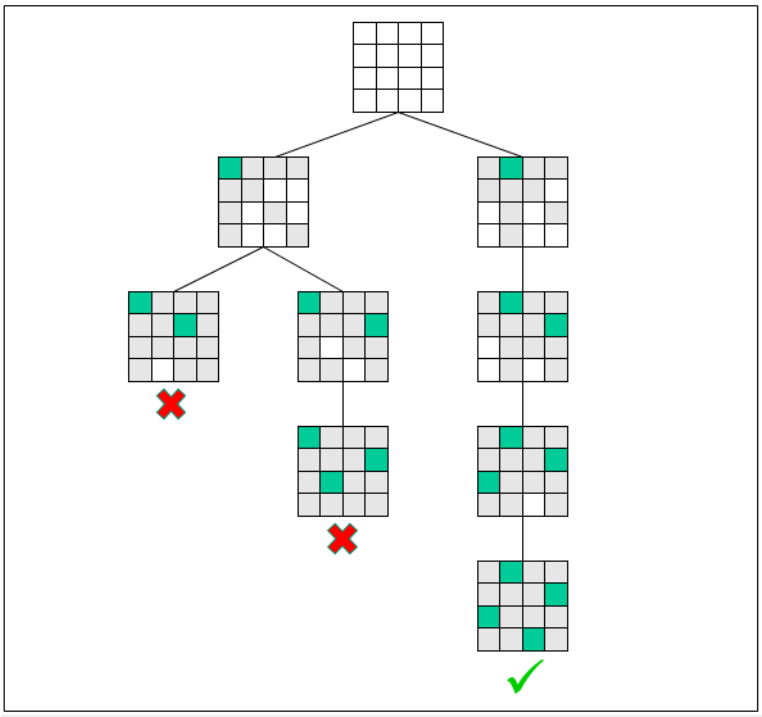

Figure 3.10 Arbre de recherche du Forward Checking

b) Algorithme de full look-ahead

L'algorithme de vérification avant (forward-checking) vérifie l'arc consistance des variables courantes avec les variables futures. Il est possible de pousser plus loin ce concept de

vérification. Ce concept consiste à vérifier l'arc consistance non seulement avec la variable qui sera prochainement instanciée, mais aussi avec les suivantes. C'est une forme de maintenance de la consistance d'arc, appelée aussi full look-ahead (FLA)[110]. Cet algorithme est encore plus efficace que FC, puisqu'il permet de détecter les inconsistances futures encore plus tôt. De même qu'avec FC, il n'est pas nécessaire de vérifier la consistance avec les variables précédemment instanciées ; puisque les domaines sont réduits pour s'ajuster automatiquement aux premières variables. La figure (3.11), montre l'arbre de recherche du Full Look Ahead.

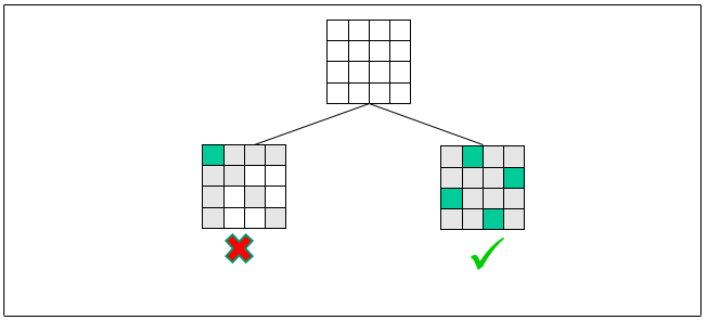

Figure 3.11 Arbre de recherche du look-Ahead

Cependant, s'il existe beaucoup de variables ayant des grands domaines, l'algorithme Full Look-Ahead fera plus de vérifications à chaque étape que l'algorithme FC.

3.4.4 Programmation quadratique

Les problèmes de satisfaction de contraintes ont été reconnus comme des modèles génériques pour résoudre plusieurs problèmes combinatoires et complexes. Dans ces dernières années, une nouvelle directive a été entamée dans le but de traiter ce type de problèmes. Cette directive consiste à modéliser ce dernier en termes d'un problème de programmation mathématique. Cette démarche a donné la naissance à une multitude d'approches exactes et heuristiques pour traiter les problèmes de type CSP. Par conséquent, la résolution de ce modèle peut être entamée par une gamme très riche de méthodes issues de la recherche opérationnelle, parmi lesquels en trouve les réseaux de neurones artificiels[52].

3.4.5 Strategies d'énumération

Un algorithme de recherche pour la satisfaction des contraintes nécessite un ordre dans lequel les variables seront étudiées et aussi l'ordre dans lequel les valeurs seront choisies.

La sélection de l'ordre correct des variables et valeurs peut considérablement améliorer l'efficacité de la résolution. Les heuristiques des variables et des valeurs jouent un rôle important dans la résolution des CSP.

Pour établir ces ordres des heuristiques de sélection de variables et des heuristiques de sélection de valeurs sont utilisées respectivement pour la résolution de CSP[105].

a) Heuristiques de selection de variables

La recherche soutient que l'ordre dans lequel les variables sont affectées au cours de la recherche affecte fortement la taille de l'espace exploré à la recherche de solutions, ce qui démontre que l'ordre des variables est crucial pour le traitement d'une solution efficace. Certaines heuristiques pour la sélection de variables permettent de diminuer le temps de calcul pendant le processus de recherche.

La littérature présente différentes voies pour le développement de cette sélection, qui dépend de la nature du problème, pour diriger une propagation efficace des contraintes. Deux types d'heuristiques d'ordre de variable doivent être pris en compte.

L'heuristique de sélection des variables statiques: qui génère un ordre fixe des variables avant de lancer la recherche, sur la base d'informations globales dérivées du graphe de contraintes initial.

L'heuristique de sélections variables dynamiques: qui peut changer l'ordre des variables dynamiquement basé sur l'information locale qui est généré pendant la recherche.

L'idée principale qui existe après le choix d'une variable est de minimiser la taille de l'arbre de recherche (espace de recherche amélioré) et de s'assurer que les branches de l'arbre qui n'ont pas de solution seront élaguées le plus rapidement possible [111].

L'ordre dans lequel les variables sont affectées lors de la recherche peut avoir un impact significatif sur la taille de l'espace de recherche. Généralement, les heuristiques de sélection de variables essaient de sélectionner les variables qui limitent le plus les autres dès que possible. L'intuition est d'essayer d'affecter le plus rapidement possible les variables les plus contraintes.

La sélection des variables peut être dynamique ou statique, les termes dynamiques et statiques se réfèrent au moment où l'ordre dans lequel les variables pour l'instanciation sont considérées.

Heuristiques de sélection statique

Ces heuristiques de sélection statiques sont basées sur les informations globales dérivées de la topologie du graphe de contraintes d'origine qui représente le CSP.

Ces heuristiques de sélection statiques génèrent un ordre fixe, des variables avant de commencer la recherche, sur la base d'informations globales dérivées de la structure du réseau de contraintes d'origine qui représente le CSP[105]. Parmi les heuristiques les plus utilisées, nous pouvons citer:

- *Largeur minimale* : considère la largeur de la variable x comme le nombre de variables qui sont avant x, selon un ordre donné, et qui sont adjacentes à x. Les variables sont ordonnées du dernier au premier en largeur décroissante. Cela signifie que les variables qui sont au début de l'ordre sont les plus contraintes et les variables qui sont à la fin de l'ordre sont les moins contraintes. Cette heuristique est particulièrement utile pour les CSP où le degré des nœuds dans le graphe de contraintes varie significativement.
- *Degré maximal* : ordonne les variables dans un ordre décroissant de leur degré dans le graphe de contraintes. Le degré d'un nœud est défini comme le nombre de nœuds qui lui sont adjacents. Cette heuristique vise également à trouver un ordre de largeur minimum, bien qu'il ne le garantisse pas.

Heuristiques de sélection dynamiques

Les heuristiques dynamiques peuvent changer l'ordre des variables dynamiquement en fonction des informations locales générées pendant la recherche. Généralement, les heuristiques de sélection dynamique essaient de sélectionner les variables qui limitent le plus les autres dès que possible. L'idée principale est d'essayer d'affecter le plus rapidement possible les variables les plus contraintes et réduisant ainsi le nombre de retour en arrière. Dans la littérature, plusieurs heuristiques de sélection dynamiques ont été proposées, les plus courantes étant: [105].

- *Valeurs restantes minimales*: consiste à sélectionner une variable avec un domaine plus petit. Cette technique est basée sur l'idée que si une variable a peu de valeurs dans son domaine, alors il est plus difficile de trouver une valeur cohérente.
- *Cardinalité maximale* : Cette heuristique, appelée Degré Max, consiste à sélectionner arbitrairement la première variable, puis à sélectionner la variable liée au plus grand nombre de variables déjà instanciées.

b) Heuristiques de sélection de valeurs

Une fois la décision d'instancier une variable est prise, plusieurs valeurs peuvent être disponibles pour la sélection. Encore une fois, l'ordre dans lequel ces valeurs sont considérées peut avoir un impact important sur le temps pour trouver la première solution.

Un ordre de valeurs permet de rechercher une branche qui mène à une solution plus tôt que les branches qui ne mènent pas à la solution. Par exemple, si le CSP a une solution, et si une valeur correcte est choisie pour chaque variable, alors une solution peut être trouvée sans retour arrière. Supposons que nous avons sélectionné une variable à instancier alors comment choisir une valeur associée à cette variable? Cette question nous ramène à traiter deux cas possibles. Dans le cas où aucune valeur ne réussit à satisfaire les contraintes, alors l'ordre de valeurs n'a pas d'importance. D'autre part, si nous pouvons trouver une solution complète basée sur les instanciations passées, alors nous pouvons utiliser une des techniques des heuristiques de sélection de valeurs[105].

- *Min-Conflits*: cette heuristique ordonne les valeurs en fonction des conflits dans lesquels elles sont impliquées avec les variables non instanciées. Chaque valeur de la variable x_i est associée au nombre total de tuples qui sont incompatibles avec les contraintes qui sont impliquées dans la variable x_i. Donc, la valeur avec la plus petite somme est sélectionnée[112].
- *Domaine de taille maximale*: est un algorithme qui consiste à sélectionner la valeur qui laisse pour les futures variables un domaine de taille maximale[112].

3.4.6 Les méthodes de décomposition structurelles

Depuis l'apparence du formalisme CSP, plusieurs méthodes ont été proposées pour résoudre efficacement ce type de problème. Toutefois, il n'existe pas actuellement une méthode universelle capable de résoudre tous les problèmes de satisfaction de contraintes d'une manière efficace, à cause de leurs complexités[113]. Pour réduire cette complexité, beaucoup de méthodes ont été menés en exploitant certaines propriétés des instances. Parmi ces méthodes, on trouve les méthodes de décomposition structurelles. Ces méthodes ont pour but de transformer un problème de satisfaction de contraintes en un autre moins complexe que l'original mais équivalent en solutions. Ces méthodes de décomposition structurelles jouent sur la structure du problème pour améliorer la résolution[113]. Cette structure est capturée à travers une représentation du problème de satisfaction de contraintes en graphe. La plupart des méthodes structurelles utilise les décompositions de graphes. Ces méthodes exploitent les

propriétés topologiques du graphe de contraintes et sont basées sur la notion de décomposition arborescente de graphes[114].

Ces méthodes structurelles proposent de meilleures garanties théoriques. Elles utilisent pour la plupart un recouvrement acyclique du problème. La résolution des différentes parties ainsi construites, donne un CSP acyclique qui peut être résolu en un temps polynomial. Les excellentes bornes de complexités de ces méthodes sont obtenues généralement au détriment de l'efficacité en pratique. En effet, il existe une inadéquation entre les bonnes bornes de complexité de ces méthodes et leurs pauvretés de performances en pratiques. L'espace mémoire requis est souvent inaccessible et rend inopérant la majorité de ces techniques. L'objectif est donc de les combiner avec des heuristiques pour le calcul des décompositions arborescentes de qualité et pour une meilleure exploitation de ces dernières, et tout cela d'un point de vue pratique. Toutefois, certaines d'entre elles, comme Backtracking sur le graphe de décomposition[97], associent ces garanties théoriques avec la souplesse de l'énumération.

3.5 Extensions de problèmes de satisfaction de contraintes

L'utilisation des problèmes de satisfaction de contraintes dans le cadre d'applications réelles met en évidence un ensemble de limitations qui ne sont pas nécessairement liés à l'inefficacité des algorithmes mais à la simplicité du formalisme, qui ne permet pas de construire une modélisation réaliste de certains problèmes. Ainsi, depuis le début des années 1990, plusieurs travaux sont intéressés à l'extension du formalisme des réseaux de contraintes. Parmi ces extensions on peut citer une multitude de type de réseau de contraintes :

– Problème de satisfaction maximale de contraintes (MaxCSP),

– Problème de satisfaction de contraintes pondérées (WCSP),

– Problème de satisfaction de contraintes symboliques (SCSP),

– Problème de satisfaction de contraintes et d'optimisation sous contraintes (CSOP),

– etc.

3.5.1 Problème de satisfaction maximale de contraintes (Max-CSP)

Parmi les problèmes de satisfaction de contraintes, il y a des problèmes qui n'ont pas de solution complète. Résoudre ces problèmes par les algorithmes vus précédemment conduira à un échec. Une solution incomplète est une solution partielle qui peut être acceptée. Il convient

alors soit d'assouplir les contraintes, soit de satisfaire le plus grand nombre de contraintes possibles. Dans le premier cas, assouplir les contraintes consiste par exemple à agrandir les domaines, supprimer une variable ou une contrainte. Dans le second cas, trouver une adéquation à toutes les variables de telle sorte que le nombre de contraintes violées soit minimal.

D'une manière naturelle lorsqu'on modélise une application ou un problème réel sous la forme d'un problème de satisfaction de contrainte, il arrive souvent qu'il n'existe aucune solution ou il est difficile de trouver cette solution. Le problème est alors qualifié de sur-contrainte. Dans cette situation, on cherche souvent à trouver un compromis : on admet une solution qui ne respecte pas toutes les contraintes. Alors, le problème sur-contraint (Max-CSP) consiste à minimiser le nombre de contraintes violées [68]. Dans ce cadre, on cherche à minimiser le nombre de contraintes violées par l'intermédiaire d'un critère d'optimisation. Alors, un problème Max-CSP est défini par un doublé (CSP, g) où :

– CSP est un problème satisfaction de contraintes.

– g est la fonction objective qui consiste à minimiser le nombre de contraintes violées.

Exemple : Soit le problème Max-CSP, P = (CSP, g) tel que :

CSP est un problème de satisfaction de contraintes défini par :

- $X = \{ x_1, x_2, x_3\}$ est l'ensemble des 3 variables,
- $D = \{ D(x_1), D(x_2), D(x_3)\}$ $tel\ que\ D(x_1) = D(x_2) = D(x_3) = \{1,2\}$
- $C = \{ C_{12}, C_{13}, C_{23}\}$ est l'ensemble de 3 contraintes tel que chaque contrainte C_{ij} est définie par $x_i = x_j$.
- $R = \{ R_{12}, R_{13}, R_{23}\}$ est l'ensemble de 3 relations tel que chaque relation R_{ij} contient les couples suivants:$\{(1,2), (2,1)\}$.
- g est la fonction objectif qui consiste à minimiser le nombre de contraintes violées. Le problème consiste à trouver une solution minimisant le nombre de contraintes violées (Figure 3.12).

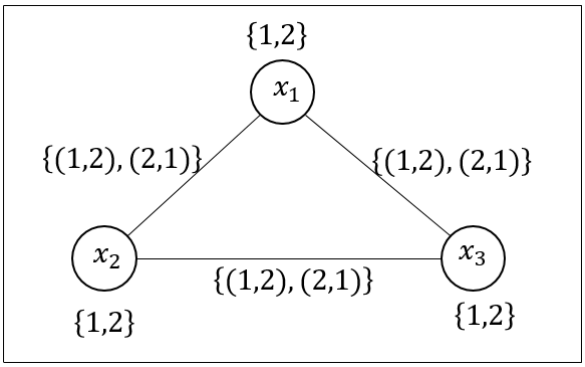

Figure 3.12 Exemple de problème de satisfaction de contraintes maximales

Ce problème n'admet pas une solution qui satisfait la totalité des contraintes. Alors on peut chercher un autre type de solutions qui consistent à affecter des valeurs aux variables du problème tout en satisfaisant le maximum de contraintes, cela revient à minimiser la fonction objective g.

Il existe plusieurs approches permettant de résoudre les problèmes MaxCSP, parmi eux en trouve les approches systématiques et les approches approchées. Les méthodes exhaustives comme séparation et évaluation (Branch and Bound B&B) donnent la solution optimale. La méthode heuristique basée sur une recherche locale donne une solution incomplète qui n'est pas forcement optimale.

- **Méthodes de resolution des problèmes Max-CSP**

Le problème MaxCSP consiste à identifier une instanciation complète qui maximise le nombre de contraintes satisfaites. Les algorithmes de résolution du problème MaxCSP sont généralement formulés pour suivre un schéma de recherche de type séparation et évaluation [115], [116]. Le principe commun à ces algorithmes consiste à calculer une borne inferieure du nombre de contraintes violées. Cependant, ces algorithmes sont connus par une efficacité faible et seuls des problèmes de tailles très réduites peuvent être résolus dans un temps raisonnable[117].

Dans ce cadre une série des techniques sont développées dans le but de calculer une meilleure borne inferieure pour ce type de problème. Ces techniques consistent à employer un mécanisme de séparation et évaluation, explorant l'espace de recherche en profondeur d'abord tout en maintenant une borne supérieure. Le coût de la meilleure solution trouvée minorant la meilleure extension possible de l'instanciation partielle courante. Ce coût représente une borne inferieure. En effet, lorsque la valeur de la borne inferieure devient plus

grande ou égale à la valeur de borne supérieure, un retour-arrière ou un processus de filtrage s'impose. Dans ce contexte, une multitude d'algorithmes sont élaborés, parmi eux il y a :

– Partial Forward Checking : L'algorithme de résolution Partial Forward Checking(PFC) est l'un des premiers algorithmes utilisés pour résoudre les problèmes Max-CSP. Il se base sur les algorithmes de séparation et évaluation. Le PFC utilise un minorant simple à calculer qui représente le nombre de contraintes violées dans le réseau. Ce minorant est utilisé pour réduire les domaines des variables[107].

– Partial Forward Checking et consistance d'arc orientée: Le minorant calculé par l'algorithme Partial Forward Checking ne prend pas en compte les violations de contraintes entre variables futures. Dans ce cadre, une amélioration de la qualité de ce minorant par l'ajout d'une évaluation du nombre de contraintes violées dans les variables futures est proposée[118].

– Partial Forward Checking et consistance d'arc orientée réversible : La méthode proposée dans ce cadre se base sur l'orientation des contraintes. Une amélioration est proposée pour la borne obtenue par l'orientation des contraintes portant sur des variables futures par l'ajout de contributions globales de sous-ensembles (Partition-Based Lower Bound)[119]. Il existe maintenant des techniques de consistances locales plus élégantes et plus puissantes[120].

– Bornes inferieures à base d'inégalités pour les problèmes Max-CSP : Dans le cadre de l'optimisation du problème Max-CSP, une exploitation de la notion d'inégalité est utilisée dans le but de calculer une borne inferieure. Cette technique peut être exploitée pour construire des modèles linéaires utiles pour le calcul de bornes inferieures. Ce calcul d'une borne inferieure à base de cliques est réalisé par une recherche locale[121].
Dans cette section, nous avons présenté le problème de satisfaction maximale de contraintes ainsi que les méthodes de résolution existantes dans la littérature. Ce problème et connu par sa forte complexité. Nous avons proposé un modèle de programmation mathématique dans le but de résoudre efficacement ce type de problème.

3.5.2 Problème de satisfaction de contraintes et d'optimisation sous contraintes (CSOP)

Dans de nombreuses applications du monde réel, il est nécessaire de trouver la meilleure solution possible à un problème et pas seulement une solution qui satisfait toutes les

contraintes. C'est ce qui justifie l'extension du cadre CSP au cadre CSOP via l'introduction d'une fonction de coût dont la valeur dépend des valeurs attribuées aux variables.
Par rapport aux CSP qui considèrent toutes les solutions possibles comme de bonnes solutions, les CSOP explorent toutes ces solutions, évaluent leurs coûts respectifs par rapport à une fonction objective appelée coût et ne retiennent que celles offrant un coût optimal. Dans le cas de Max-CSP, le coût - également appelé fonction objectif - consiste à maximiser le nombre de contraintes satisfaites. Dans ce qui suit, nous définissons le formalisme CSOP ainsi que les différentes techniques de résolution qui y sont associées[122].

- **Formalisme**

Formellement, un CSOP est un quintuple (X, D, C, R, f) défini par:

- $X = \{ x_1, \dots, x_n\}$ est un ensemble de n variables.
- $D = \{ D(x_1), \dots, D(x_n)\}$ est un ensemble de définition pour chaque variable.
- $C = \{ C_1, \dots, C_m\}$ est un ensemble de m contraintes. Chaque contrainte définie la relation existante entre certaines variables, restreignant les valeurs que peuvent prendre simultanément ces variables.
- $R = \{ R_1, \dots, R_m\}$ est un ensemble de m relations, telle que chaque relation R_i est associée à une contrainte C_i. La relation R_i est définie sur $D(x_i) \times D(x_j)$, représentant les affectations possibles qui peuvent être prise par les variables tout en respectant la contrainte C_i.
- $f: S_p \rightarrow valeur\ numerique$ est une fonction de coût ou d'objectif, où S_p est l'ensemble des solutions potentielles.

Ainsi, un $CSOP(X, D, C, R, f)$ est un $CSP(X, D, C, R)$ qui recherche des instanciations complètes et cohérentes optimisant la fonction de coût f.

- **Exemple**

  A titre d'exemple, nous modifions l'exemple de placement des reines en ajoutant une fonction objectif f qui renvoie une somme des valeurs de trois premières variables. Par conséquent, nous pouvons écrire :

$$f(\{x_1 = v_1\ , x_2 = v_2\ , x_3 = v_3\}) = \sum_{i=1}^{3} v_i$$

Le problème devient maintenant un problème de CSOP. Dans l'exemple de placement des reines, il y a deux solutions $\{x_1 = 2,\ x_2 = 4,\ x_3 = 1\ et\ \ x_4 = 3\}\ et\ \{x_1 = 3,\ x_2 = 1,\ x_3 = 4\ et\ \ x_4 = 2\}$. La fonction objective de la première solution est de coût égal 7, et la seconde est de coût égal 8. Si le CSOP est un problème de minimisation. La première solution est la solution optimale. Les méthodes de résolution peuvent être classées en deux grandes catégories: les méthodes systématiques qui garantissent l'exhaustivité de la résolution mais exigent beaucoup de temps. Les méthodes approximatives sacrifient l'exhaustivité pour gagner en complexité temporelle:

- Les méthodes systématiques[121]: Malgré la qualité optimale de leurs résultats, ces approches rencontrent généralement des difficultés avec des applications de grande taille. Dans cette catégorie, nous discutons de la méthode la plus connue comme séparation et évaluation. Ces méthodes systématiques définies dans le cadre CSP peuvent facilement être adaptées au cas des CSOP.

- Méthodes approchées[123]: Ces méthodes sont particulièrement utilisées par les chercheurs qui traitent des problèmes d'optimisation à grande échelle nécessitant une résolution rapide et une optimalité optimale.

3.5.3 Problème de satisfaction de contraintes pondérées(WCSP)

Dans de nombreux domaines d'application, les contraintes ne sont pas définies au sens strict (en d'autre terme ne sont pas des contraintes dures) pour modéliser un problème donné. De nombreux extensions ont étendus les CSP pour modéliser des préférences pour certains tuples de valeurs, ou encore des coûts de violation (notion de contrainte souple) comme par exemple les réseaux de contraintes floues, les réseaux de contraintes probabilistes.
Un WCSP [64]– [66] est une version d'optimisation d'un CSP dans lequel les contraintes ne sont plus "dures", mais sont étendues en associant des coûts non négatifs aux tuples. Le but est de trouver une attribution de valeurs à toutes les variables de leurs domaines respectifs de sorte que le coût total soit minimisé. Les WCSP fournissent un outil puissant pour modéliser des problèmes pertinents, y compris les problèmes d'allocation[83].

- **Formalisme**

Plus formellement, un WCSP est défini par un quintuple (X, D, C, R, W) défini par:

- $X = \{ x_1, \dots, x_n\}$ est un ensemble de n variables.

- $D = \{ D(x_1), \dots, D(x_n)\}$ est un ensemble de définition pour chaque variable.
- $C = \{ C_1, \dots, C_m\}$ est un ensemble de m contraintes. Chaque contrainte définie la relation existante entre certaines variables, restreignant les valeurs que peuvent prendre simultanément ces variables.
- $R = \{ R_1, \dots, R_m\}$ est un ensemble de m relations, telle que chaque relation R_i est associée à une contrainte C_i. La relation R_i est définie sur $D(x_i) \times D(x_j)$, représentant les affectations possibles qui peuvent être prise par les variables tout en respectant la contrainte C_i.
- W : est un ensemble de contraintes pondérées (c'est-à-dire des fonctions de coût). Chaque $w \in W$ est défini sur un sous-ensemble de variables. La fonction objectif est la somme de toutes les fonctions de W, $W = \sum_{w \in W} w$ et le but est de trouver l'instanciation des variables qui minimise la fonction objectif.
- **Exemple**

Dans cet exemple, nous présentons un problème de satisfaction de contraintes pondérées. Dans ce problème, nous avons deux variables $x \; et \; y$, définies sur le domaine $D = \{1, 2, 3, 4\}$. Dans ce type de problème, nous avons un ensemble de fonctions de coût définies sur les variables d'intérêt; chaque fonction de coût décrit le coût d'une suite spécifique d'une configuration sous une instanciation particulière des variables. Pour simplifier, nous définissons un coût de fonction $cout(a, n) = (a - n)^2$. En particulier, nous allons définir trois contraintes notées $C_x, C_y \; et \; C_{x,y}$ définies comme suit:

$$C_x = cout(2, x)$$
$$C_y = cout(4, y)$$
$$C_{x,y} = cout(1, y - x)$$

Les contraintes unaires $C_x \; et \; C_y$ sont destinées à représenter les coûts associés à une instanciation en cherchant une valeur idéale. Par exemple, la valeur idéale pour x selon C_x est 2 et toute instanciation où x n'est pas fixée à cette valeur sera pénalisée proportionnellement au carré de sa distance par rapport à cette valeur. La contrainte binaire $C_{x,y}$ est utilisée pour illustrer la modélisation des relations complexes entre des instanciations de variables.

En particulier, nous démontrons l'évaluation de la combinaison des contraintes $C_x, \; C_y \; et \; C_{x,y}$ sous l'instanciation où x à la valeur 1 et y à la valeur 5.

$$C_x[x \coloneqq 1, y \coloneqq 5] = cout(2,1) = 1$$

$$C_y[x \coloneqq 1, y \coloneqq 5] = cout(4,5) = 1$$

$$C_{x,y}[x \coloneqq 1, y \coloneqq 5] = cout(1,5-1) = 9$$

Le coût associé à cette instanciation des variables égal 11.

3.6 Conclusion

Le problème de satisfaction de contraintes offre un cadre puissant pour la représentation et la résolution efficace de nombreux problèmes. En général, un problème CSP comporte un certain nombre de variables, chacune ayant un domaine fini, et un certain nombre de contraintes sachant qu'une contrainte implique une ou plusieurs variables définissant la relation entre ces dernières. Trouver une solution à un problème CSP consiste à affecter une valeur à chaque variable, de telle sorte que la totalité des contraintes soient satisfaites. Lorsqu'une contrainte implique deux variables, on parle de contrainte binaire, mais quand elle implique un nombre non déterminé de variables, on parle de contrainte n-aire. Pour résoudre ces problèmes CSP, plusieurs approches ont été proposées dans ce cadre. Certains d'entre elles utilisent la propagation des contraintes pour simplifier le problème original et accélèrent ainsi le processus de résolution. D'autres utilisent l'algorithme retour arrière pour chercher des solutions possibles. D'autre combinent ces deux techniques précédentes. De plus, un modèle de programmation quadratique représentant le problème de satisfaction maximale de contraintes a été proposé. Ce modèle a donné naissance à plusieurs algorithmes de résolution issue de la programmation mathématiques.

Dans ce chapitre nous avons introduit le problème de satisfaction de contraintes. Par la suite, quelques problèmes classiques qui peuvent être représenté comme un problème CSP sont étudiés. Nous avons également décris les principales méthodes de résolution utilisées actuellement. Puis, nous avons étudié quelques extensions des problèmes de satisfaction de contraintes à savoir les problèmes de satisfaction maximale de contraintes, problème d'optimisation de contraintes et problème de satisfaction de contraintes pondérées.

Chapitre 4 : Modélisation, simulation et analyse d'expériences

4.1 Introduction

Nous avons abordé dans le premier chapitre la problématique d'optimisation des requêtes en bases de données et de la programmation par contraintes qui représente un défi en informatique. En se focalisant sur cette thématique, nous avons présenté les principales techniques d'optimisation ainsi les travaux élaborés dans ce domaine. Nous avons présenté dans le deuxième et troisième chapitre les approches récentes et efficaces que nous adoptons dans ce chapitre pour la résolution de la problématique posée.

Ce chapitre présente le vif de nos travaux de recherche, il précise l'environnement de test utilisé dans les expériences réalisées et décrit les évaluations comparatives effectuées sur les algorithmes d'optimisation. Nos contributions sont présentées dans trois parties. La première partie est consacrée à analyser et mettre en œuvre l'approche proposée pour optimiser la performance de requêtes sur des données massives dans un environnement centralisé afin d'augmenter la vitesse d'exécution des requêtes.

Pour présenter l'approche proposée, nous commençons par la représentation de l'espace de recherche à partir d'un ensemble des requêtes. Ce dernier est modélisé sous la forme de problème de satisfaction de contraintes pondérées dans le but d'étudier l'influence de la variation des pondérations sur la vitesse d'exécution. A base de cet espace, nous adoptons la technique de matérialisation via l'algorithme génétique en étudiant les différents coûts de traitement, maintenance et de stockage pour accélérer l'exécution des requêtes.

La deuxième partie vise à analyser le problème de stable maximum, et mettre en œuvre l'approche proposée pour évaluer la performance de système continu de Hopfield amélioré. En effet, cette partie est dédiée à la formulation quadratique du problème du stable maximum

et la modélisation de celui-ci sous forme d'un problème de Hopfield amélioré. Contre la faiblesse du réseau de Hopfield classique, nous avons adopté une nouvelle méthode d'Adams bashforth afin de proposer une nouvelle discrétisation de l'équation différentielle caractérisant le réseau de Hopfield.

Dans la troisième partie, nous proposons une nouvelle approche pour résoudre les problèmes binaires de satisfaction maximale de contraintes (Max-CSP). Dans cette nouvelle approche, le problème à résoudre est décrit en termes d'un problème de programmation quadratique sous des contraintes linéaires. Ce problème a été résolu via l'algorithme génétique.

4.2 Simulation du problème de sélection de vues à base de TPC-H

Le problème de sélection de vues consiste à choisir un sous-ensemble de vues pour la matérialisation afin d'améliorer la performance de requêtes sur des données massives dans un environnement centralisé pour augmenter la vitesse d'exécution des requêtes. En générale, le problème de sélection de vues implique de minimiser le coût total. Le problème de sélection de vues est un problème NP-difficile[9].

Dans cette partie, nous mettons en œuvre l'approche de matérialisation pour la sélection d'un sous-ensemble optimal de vues. Dans ce problème de sélection de vues, tous les coûts: le coût de traitement de requêtes, le coût de maintenance et le coût de stockage sont utilisés pour améliorer la performance de requêtes afin d'augmenter la vitesse d'exécution des requêtes.

4.2.1 Description de benchmark TPC-H

Un benchmark est un moyen de mesurer la performance d'un système informatique. En outre, nous définissons le benchmark comme l'ensemble de données et de tests utilisés pour évaluer les performances et la qualité d'une machine[21].

Le benchmark TPC-H offre un ensemble de données et des requêtes conçues spécifiquement pour représenter un vaste secteur industriel, tout en conservant un degré suffisant de facilité de mise en œuvre. L'utilisation de ce benchmark apporte deux avantages principaux:

1. Capacité d'examination de grandes quantités de données. En fait, le benchmark offre la flexibilité nécessaire pour mettre en place une base de données massive ayant une taille approximative de 1Go à 100.000Go sur laquelle nous pouvons effectuer nos expériences. Dans notre cas, nous avons construit une base de données de test faisant une taille d'un (1) Go.

2. Exécuter des requêtes avec un haut degré de complexité, au moins supérieur à celles généralement utilisées dans les systèmes transactionnels, permettant ainsi l'utilisation de tous les opérateurs disponibles dans le système de gestion de base de données. Nous pouvons ainsi analyser leur exécution. D'autre part, la complexité et la grande quantité de données offertes par le benchmark donne la possibilité de stresser le serveur de base de données.

Au niveau conceptuel, la base de données de test utilisée par le benchmark est représentée dans la figure (4.1) ci-dessous.

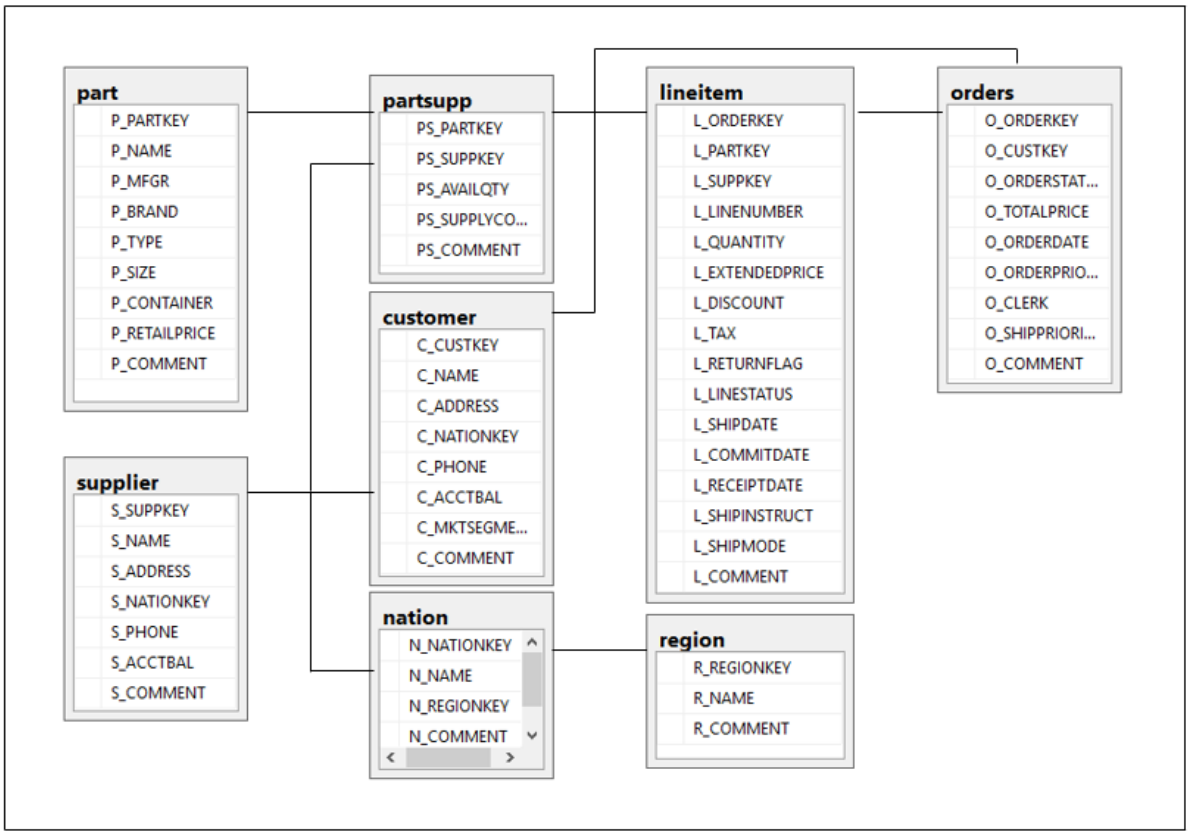

Figure 4.1 Le schéma TPC-H

Parmi les huit entités du modèle TPC-H, les deux plus grandes sont Lineitem et Orders qui peuvent être considérées comme la table de faits d'un entrepôt de données, car elles contiennent les données détaillées du modèle. La principale partie de la taille de l'entrepôt de données est conservée entre ces deux tables. Une attention particulière a été accordée à ces deux tables pendant la phase expérimentale de la recherche. Les autres tables peuvent être considérées comme des dimensions et sont petites par rapport aux tables Lineitem et Orders.

Le tableau 4.1 récapitule le nombre d'enregistrements et la taille de chaque table de la base de données utilisée par le TPC-H :

Nom de table	**Nombre des enregistrements**	**Taille de table (Mb)**
Region	5	<1
Nation	25	<1
Customer	150.000	26
Supplier	10.000	2
Part	200.000	30
Partsupp	800.000	110
Lineitem	6.000.000	641
Orders	1.500.000	149

Tableau 4.1 Nombre et taille des tables de schéma TPC-H

4.2.2 Plan d'exécution d'une requête

Un plan d'exécution pour une requête est composé d'une série d'opérations qui doivent être exécutées dans un certain ordre pour la résoudre. Nous devons prendre en compte certains des opérateurs d'algèbre relationnelle présents dans le plan d'exécution[4], [31], [128], [129]:

- Projection (π): Cet opérateur permet d'extraire des colonnes entières d'une table (relation). Il est généralement utilisé comme l'un des derniers opérateurs lors de l'exécution d'une requête, puisqu'il ne permet de renvoyer que les attributs recherchés.
- Sélection (σ): Cet opérateur renvoie les n-uplets d'une relation conforme au prédicat spécifié.
- Jointure (⋈): Il s'agit d'un opérateur basé sur la combinaison de deux relations différentes pour un attribut commun. De manière plus générale, une jointure entre deux relations peut être définie comme un produit cartésien précédé d'une sélection à condition que les attributs communs à la relation soient égaux. Il est à noter que la jointure est l'opérateur le plus cher de l'algèbre relationnelle, donc son utilisation a un grand impact sur les performances lors de l'exécution de requêtes.

En général, les plans d'exécution sont représentés par un arbre binaire ayant une relation directe avec l'algèbre relationnelle associée à la requête [29]. Chaque feuille de l'arbre est une relation et chaque nœud correspond à un opérateur. Lorsque l'exécution d'une requête se produit, l'arbre est traversé des feuilles à la racine. Chaque opérateur (nœud) est exécuté en traitant ses enfants (feuilles), le résultat de l'exécution devient une nouvelle feuille servant d'entrée au nœud parent immédiat, le traitement se poursuit ainsi jusqu'à atteinte de la racine.

La forme de l'arbre peut être différente selon le système de base de données qui le génère, dans ce travail, nous utilisant les arbres binaire gauche.

La figure (4.2) ci-dessous représente une requête sur les relations Part, Partsupp, Supplier, Nation et Region. La requête renvoie le coût d'approvisionnement minimum de chaque pays dans la région d'Asie.

```
SELECT
        N_Name, Min(Ps_Supplycost)
FROM
        Part, Partsupp, Supplier, Nation, Region
WHERE
        P_Partkey = Ps_Partkey
        AND S_Suppkey = Ps_Suppkey
        AND S_Nationkey = N_Nationkey
        AND N_Regionkey = R_Regionkey
        AND R_Name = 'ASIA'
GROUP BY
        N_Name;
```

Figure 4.2 Requete du schema TPC-H

En général, le compilateur SQL du système de gestion de base de données (SGBD) traduit la requête en un plan d'exécution. Dans notre cas, lorsque le SGBD de SQL server reçoit la requête de la figure (4.2), il génère un plan d'exécution. Les nœuds construits sont exprimés dans la figure (4.3).

Dans le cas du plan d'exécution d'un plan de la figure (4.4), le SGBD peut par exemple commencer à exécuter les opérateurs de sélection, ceci dans le but de réduire le nombre de lignes à traiter par colonnes impliqué dans les jointures. Ensuite, il exécutera les jointures et autres opérateurs nécessaires à la résolution de la requête.

```
tmp1 = selection(r_name,'ASIA','=')
tmp2 = projection(r_regionkey,tmp1)
tmp4 = projection(n_regionkey,n_nationkey,n_name,nation)
tmp3 = jointure(tmp2,tmp4)
tmp6 = projection(s_nationkey,s_suppkey,supplier)
tmp5 = jointure(tmp3,tmp6)
tmp7 = projection(ps_suppkey,ps_partkey,ps_supplycost,partsupp)
tmp9 = jointure(tmp5,tmp7)
tmp8 = projection(p_partkey,part)
tmp10 = jointure(tmp9,tmp8)
```

Figure 4.3 Nœuds construit par le SGBD de SQL Server

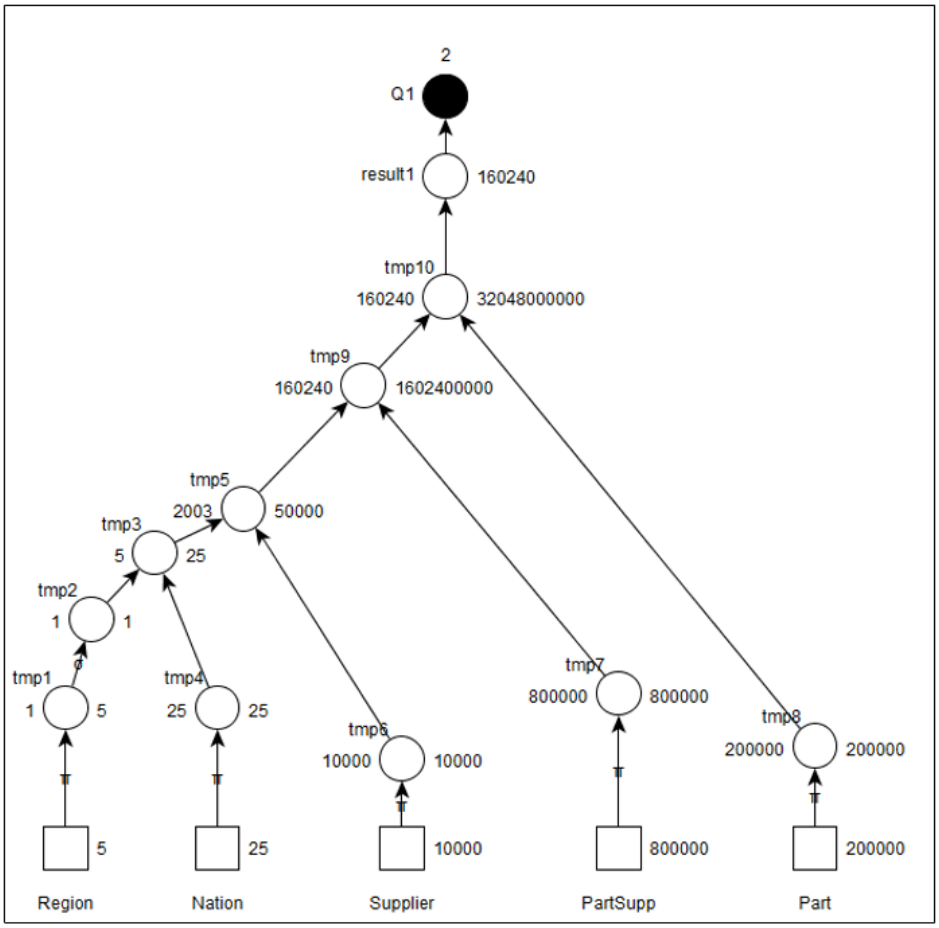

Figure 4.4 Plan d'exécution de la requête

La décomposition de la requête en un ensemble de nœuds permet de construire le plan d'exécution. Chaque requête peut avoir plusieurs plans d'exécution. Le plan global peut être obtenu à partir de plusieurs requêtes afin d'avoir un coût minimal.

4.2.3 Implémentation de l'espace de recherche

Dans le cas des multi-requêtes, les requêtes sont optimisées et exécutées par lots. Les requêtes individuelles sont transformées et représentées en un plan d'exécution (espace de recherche).

L'espace de recherche pour le problème d'optimisation des multi-requêtes est construit par un ensemble des requêtes dont chacune a des possibles plans d'exécution. Cet espace est obtenu par la fusion de tous les plans, en prenant un seul plan par requête afin d'avoir un espace optimal. Dans notre expérience, nous avons construit l'espace de recherche optimal obtenu par huit requêtes présentées dans la figure (4.5).

Toutes les opérations de jointure, sélection et projection sont représentées par des nœuds intermédiaires de manière à ce que soit chaque nœud est étiqueté par deux valeurs, la valeur

de gauche représente le coût de traitement de la requête et la valeur de droite représente le coût de maintenance[130].

Les racines correspondent aux requêtes, les feuilles schématisent les relations. Chaque requête est étiquetée par une fréquence.

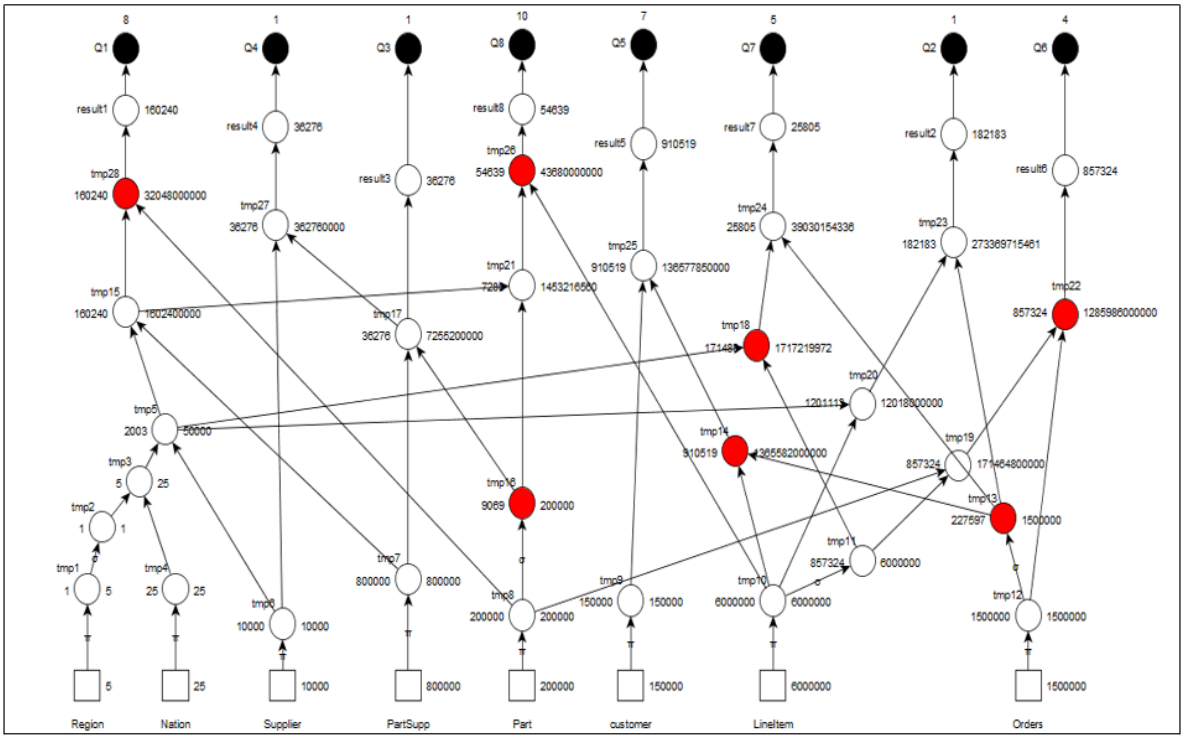

Figure 4.5 Plan multiple d'exécution de vues avec la matérialisation

L'algorithme génétique est appliqué pour sélectionner l'ensemble des vues à matérialiser sur un espace de recherche optimisé. La figure (4.5) indique que tmp28, tmp26, tmp16, tmp18, tmp14, tmp22, tmp13 sont des vues matérialisées.

NB : Toutes les requêtes, de benchmark TPC-H, utilisées dans ce travail sont présentées dans l'annexe.

4.2.4 Modélisation par contraintes de problèmes de sélection de vues

Le problème de satisfaction des contraintes pondérées (WCSP)[117] est un problème de satisfaction de contraintes (CSP) dans lequel il est possible d'exprimer des préférences dans les solutions. Au cours des dernières années, le cadre CSP a été étendu aux contraintes douces pour exprimer les préférences parmi les solutions[117], [131], [132]. Dans ce travail, le problème de sélection de vues est modélisé sous forme d'un problème de satisfaction de contraintes pondérées.

Un certain nombre de paramètres, à savoir : les coûts de traitement et fréquences des requêtes, les coûts de maintenance, les fréquences de mise à jour et les coûts de stockage, sont utilisés pour sélectionner un ensemble approprié de vues à matérialiser. Premièrement, pour transformer le problème de sélection des vues en termes de WCSP, nous définissons les concepts suivants:

- R: est l'ensemble des relations de base.
- $Q = \{q_1, q_2, \dots, q_n\}$: est l'ensemble des requêtes sur un entrepôt de données donné;
- f_q: est la fréquence de requête associée à la requête q;
- M: est l'ensemble des nœuds intermédiaires matérialisés;
- S: est le coût de stockage;
- S_v: est le coût de stockage correspond à la vue v;
- f_u: est la fréquence de mise à jour associée à la vue v.
- **Variables**: les variables du problème de satisfaction des contraintes pondérées sont données:
- $Mat(v)$: est une variable de décision égale à 1 si la vue est matérialisée, 0 sinon;
- $Cost_q(v)$: est le coût de la requête qui correspond à la vue v;
- $Cost_m(v)$: est le coût de la maintenance.
- **Domaines**: le domaine d'une variable donnée, désigné par l'ensemble des valeurs que cette variable peut prendre.

Comme la variable $Mat(v)$ est une variable binaire, alors $D(Mat(v)) = \{0,1\}$.

Les variables $Cost_q(v)$ et $Cost_m(v)$ ont pour domaines $D(Cost_q(v)) \subset \mathrm{N}$ et $D(Cost_m(v)) \subset \mathrm{N}$

- **Contraintes**: les contraintes sont divisées en deux groupes :

Contraintes binaires: La contrainte entre les variables est présentée comme suit:

Coût de requête:

$$Cost_q(v) = \begin{cases} f_q \times (le\ co\hat{u}t\ du\ chemin\ de\ q\ \text{à}\ v)\ si\ Mat(v) = 1 \\ f_q \times (le\ co\hat{u}t\ de\ l'arbre\ de\ racine\ q)\ sinon \end{cases}$$

Coût de maintenance:

$$Cost_m(v) = \begin{cases} f_u \times (le\ co\hat{u}t\ du\ chemin\ de\ v\ vers\ les\ relations\ R)\ si\ Mat(v) = 1 \\ 0\ sinon \end{cases}$$

Contraintes unaires

La valeur de la variable $Cost_m(v)$ est minimisée par une contrainte unaire avec une pondération β: $(la\ valeur\ de\ la\ variable\ de\ maintenance) \times \beta\ avec\ 0 \leq \beta \leq 1$. Le coût total de traitement des requêtes est :

$$Cost_T = \sum_{q \in Q} Cost_q(v) + \sum_{m \in M} Cost_m(v)\ st\ \sum_{v \in M} S_v \leq S.$$

4.2.5 Analyse des résultats

Les résultats obtenus dans ce travail sont réalisés en utilisant la base de données TPC-H avec un facteur d'échelle égal 1 Go ainsi que les requêtes de TPC-H comme charge de travail. Pour rappel, les composants de la base de données sont constitués de huit tables comprenant Part, Supplier, Customer, Orders, Nation, Region, Partsupp et Lineitem. Le TPC-H est utilisé ici comme moyen de mesurer la performance du système.

Le plan multiple d'exécution des vues proposé dans cette étude est un espace de recherche pour sélectionner un ensemble optimal de vues à matérialiser. Dans ce cadre, nous avons proposé une approche basée sur l'algorithme génétique qui tire sa force de la création d'une population de candidats, et une autre basée sur l'algorithme d'escalade.

L'analyse du fonctionnement de ces deux approches permet de voir une grande différence entre les deux. L'approche génétique commence par la création d'une population de solutions ce qui donne d'abord une diversité de choix et une convergence plus au moins efficace. Contrairement à l'algorithme d'escalade qui lui se base sur un seul candidat. À chaque itération, l'algorithme d'escalade essaie d'améliorer la solution précédente. Les expériences ont été menées afin de trouver un compromis entre le coût de traitement des requêtes et le coût de maintenance en utilisant ces deux algorithmes.

	Coût de requête	Coût de maintenance	Coût total
Toutes les vues virtuelles	17585777978944	0	17585777978944
Toutes les vues matérialisées	12014999	10995789073264	10995801088263
Ensemble optimal par approche génétique (AG)	1451491204595	7559210580088	9010701784683
Ensemble optimal par algorithme d'escalade (AE)	1451508620875	7902161180088	9353669800963

Tableau 4.2 Coût de requête, coût de maintenance et coût total par AG et AE

Le tableau (4.2) montre les coûts obtenus avec et sans la matérialisation et aussi par l'utilisation de deux approches génétique et l'approche d'escalade. Le coût total obtenu par l'approche génétique correspond à la matérialisation des vues tmp28, tmp26, tmp16, tmp18, tmp14, tmp22, tmp13 représentées dans la figure (4.5).

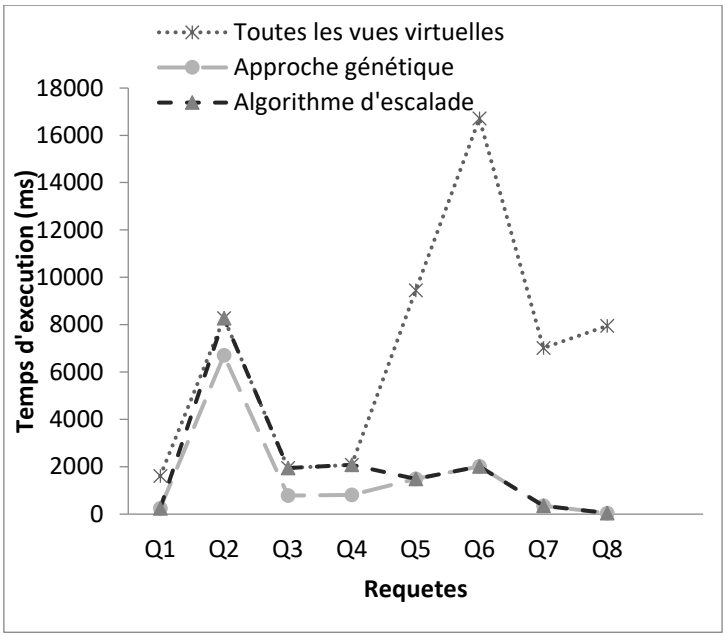

Figure 4.6 Temps d'exécution des requêtes en (ms)

La comparaison entre les deux approches pour un ensemble de requêtes révèle un résultat intéressant: le temps d'exécution sans sélection de vues matérialisées supplémentaires est long, mais avec la présence de vues matérialisées, le temps d'exécution est court. L'approche génétique donne un meilleur temps d'exécution par rapport à l'algorithme d'escalade. Il est important de mentionner que toutes les requêtes ont bénéficié de vues matérialisées.

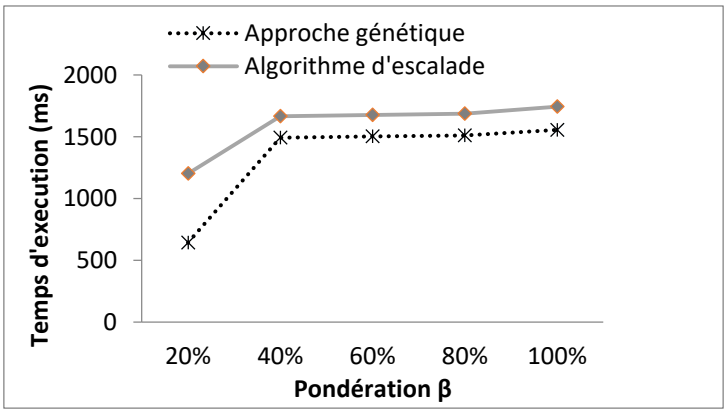

Figure 4.7 Comparaison du temps moyen d'exécution

La figure (4.7) montre le temps moyen d'exécution avec l'approche génétique et l'algorithme d'escalade en fonction des pondérations accordées ($\beta = \{20\%, 40\%, 60\%, 80\%, 100\%\}$). Pour évaluer les deux approches nous avons décidé d'adopter un ensemble de requêtes pour calculer le temps d'exécution moyen tout en changeant la pondération. A partir de la même figure, il y a une forte dépendance entre le temps d'exécution moyen et la pondération.

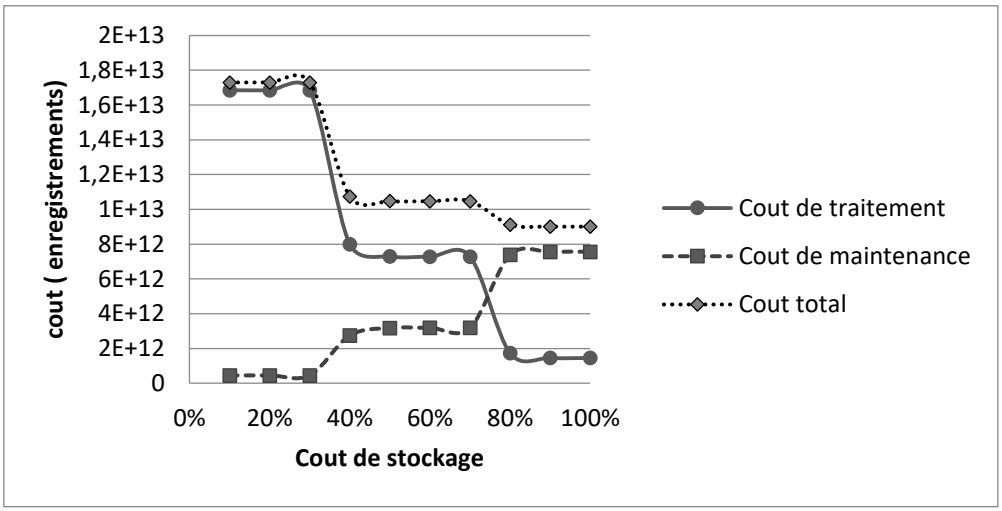

Figure 4.8 Coût de traitement et de maintenance envers le coût de stockage

L'analyse des coûts de stockage a pour objectif de trouver un compromis entre le coût associé au traitement des requêtes et le coût associé à la maintenance. La figure (4.8) représente le comportement de coût de traitement de requêtes, de maintenance et total en fonction de coût de stockage, avec le coût total correspond à la somme de deux coûts : coût associé au traitement des requêtes, celui-ci comprend les requêtes qui ont bénéficiées ou non de la matérialisation. Et le coût de maintenance c'est-à-dire celui qui est associé à la mise à jour des vues matérialisées. D'après la figure (4.8), nous remarquons que le coût total diminue lorsque le coût de stockage augmente.

4.3 Simulation du problème du stable maximum

En théorie des graphes, un ensemble indépendant ou stable est un ensemble de sommets dans un graphe tel qu'aucun d'entre eux n'est adjacent à un autre. C'est-à-dire, un ensemble (S) est dit stable si chaque arête du graphe contient au plus un sommet dans S. La taille d'un

ensemble indépendant est le nombre de sommets qu'il contient. Un ensemble est dit maximal si l'ajout d'un seul sommet perturbe l'indépendance de cet ensemble.

Dans ce paragraphe, nous formulons le problème de stable maximum comme un problème quadratique afin d'avoir une transformation vers le système continu de Hopfield. Aussi, Nous proposons la méthode d'Adams-bashforth pour discrétiser le système continu de Hopfield dans le but d'améliorer ce système.

4.3.1 Formulation mathématique de problème du stable maximum

Etant donné un graphe $G = (V, E)\ avec\ V = \{v_1, v_2, \dots, v_n\}$. Un ensemble $S \subset \mathrm{V}$ est dite stable si ses éléments ne sont pas deux à deux adjacents. Le problème de l'ensemble stable de taille maximale est celui qui permet de déterminer l'ensemble stable de taille maximale α (G). Le problème de l'ensemble stable de taille maximale est NP-difficile même pour l'approximation sauf pour des graphes qui vérifient certaines conditions particulières[133], [134]. Différentes approches ont été proposées dans la littérature pour la résolution de ce problème d'une manière systématique[135], [136].

Pour résoudre le problème de l'ensemble stable de taille maximale via le réseau de Hopfield, il faut qu'il soit modélisé sous forme d'un problème d'affectation quadratique avec contrainte mixtes. Soit $S \subset \mathrm{V}$ un ensemble stable, on définit la variable de décision x comme suit :

$$x_i = \begin{cases} 1\ si\ v_i \in S \\ 0\ sinon \end{cases}$$

Puisque deux nœuds adjacents $x_i\ et\ x_j$ ne doivent pas être dans l'ensemble stable S, on obtient la contrainte suivante : $x_i x_j = 0$.

Cette contrainte peut être agrégée dans:

$$h(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} x_i x_j = 0$$

Où

$$b_{ij} = \begin{cases} 1\ si\ (v_i, v_j) \in E \\ 0\ sinon \end{cases}$$

La fonction objective du problème de programmation quadratique est :

$$f(x) = -\sum_{i=1}^{n} x_i$$

Par conséquence, le problème de l'ensemble stable de taille maximale peut être exprimé comme suit :

$$(PQ) = \begin{cases} Min\ f(x) = -\sum_{i=1}^{n} x_i \\ h(x) = \sum_{i=1}^{n}\sum_{j=1}^{sc} b_{ij}x_i x_j = 0 \\ x \in \{0,1\}^n \end{cases}$$

La matrice b et la variable binaire x sont définies comme suit :

- Le coefficient b_{ij} vaut 1 si l'arc (i,j) figure sur le graphe, et il prend la valeur 0 sinon.

- La composante x_i vaut 1 si la variable x_i est choisie dans l'ensemble stable, et elle prend la valeur 0 sinon.

La contrainte suivante $x_i \in \{0,1\}$ peut être définie de la manière suivante :

$$x_i \in \{0,1\} \Leftrightarrow x_i^2 - x_i = 0$$

- **Exemple**

Considérons le graphe présenté dans la figure (4.9), ce graphe se compose de huit nœuds et neuf arcs. Ce graphe contient trois ensembles stables :

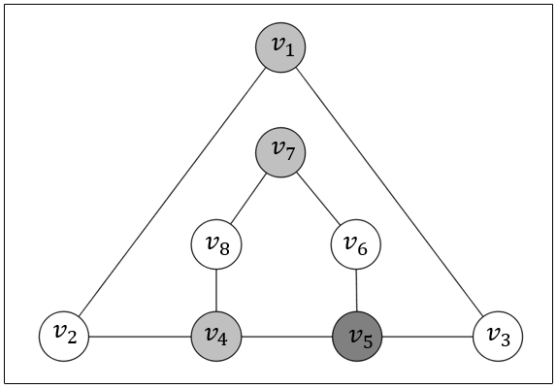

Figure 4.9 Graphe de trois ensembles stables

Cette figure peut être modélisée mathématiquement comme un problème quadratique suivant :

$$(QP)\begin{cases} Min\ f(x) = -\sum_{i=1}^{8} x_i \\ sc \\ h(x) = x^T B x \\ x \in \{0,1\}^8 \end{cases}$$

$$B = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$$

Enfin, le problème de l'ensemble stable maximal a été modélisé en termes d'un programme quadratique à variables bivalentes avec contraintes quadratiques (*QP*).

4.3.2 Système continu de Hopfield pour le problème du stable maximum

Dans cette partie, nous allons voir comment calculer un ensemble stable pour le problème de l'ensemble stable de taille maximale d'un graphe via le système continu de Hopfield[56], [58]. Il est donc nécessaire de concevoir une fonction énergétique qui répond aux particularités du problème de l'ensemble stable de taille maximale. Nous proposons la fonction d'énergie suivante :

$$E(x) = -\alpha \sum_{i=1}^{n} x_i + \frac{1}{2}\phi \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} x_i x_j + \gamma \sum_{i=1}^{n} x_i(1 - x_i)$$

Les paramètres ϕ, γ et α doivent être sélectionnés tel que tout point d'équilibre du système continu de Hopfield soit réalisable.

La procédure du réglage des paramètres est basée sur les dérivées premières de la fonction énergétique.

$$\frac{\partial E(x)}{\partial x_i} = -\alpha + \phi \sum_{j=1}^{n} b_{ij} x_j + \gamma(1 - 2x_i)$$

Au début, nous imposons les conditions $\phi > 0$ et $\gamma \geq 0$. Pour minimiser la fonction objective, nous imposons la contrainte $\alpha > 0$.

Ainsi, une solution réalisable est donnée par : $\alpha > 0$, $\phi > 0$, $\gamma \geq 0$ et $-\alpha + \phi - \gamma \geq \varepsilon$. En générale, la solution fournie par le système continu de Hopfield aboutit à un ensemble stable non maximal, Pour améliorer cette solution, nous proposons une nouvelle approche qui se base sur la discrétisation du système continu de Hopfield.

4.3.3 Discrétisation du système continu de Hopfield

Dans ce travail, nous avons dû maintenir et perfectionner le système continu de Hopfield afin de résoudre le problème du stable maximum. Nous avons aussi comparé les résultats obtenus avec les deux autres méthodes numériques appliquées à l'équation différentielle.
Le système continu de Hopfield est caractérisé par l'équation différentielle qui prend la forme générale:

$$\frac{du(t)}{dt} = f(t, u(t))$$

La discrétisation de système continu de Hopfield est souvent donnée par la méthode numérique d'Euler, qui est définie par l'équation suivante:

$$u_{n+1} = u_n + h \times f(u_n)$$

Cette méthode est très sensible à la condition initiale et à la taille de pas. De plus, la méthode d'Euler produit des solutions locales qui ne sont pas assez bonnes. Pour surmonter ce problème, nous avons proposé dans ce travail la méthode multi-pas d'Adams-Bashforth de second ordre. L'idée de cette méthode est de prendre différentes étapes dans le temps afin d'atteindre la précision ciblé[137].
Considérons l'équation différentielle caractérisant le réseau de Hopfield le long d'une étape de t_n à t_{n+1}. En intégrant de t_n à t_{n+1}, nous pouvons écrire l'équation différentielle sous la forme :

$$u(t_{n+1}) = u(t_n) + \int_{t_n}^{t_{n+1}} f\big(t, u(t)\big)dt \quad (4.1)$$

La méthode d'Euler est l'une des techniques qui permet d'approximer une équation différentielle de premier ordre. Elle rapproche l'intégrale par l'aire d'un rectangle dont la base a la longueur $(t_{n+1} - t_n)$ et dont la hauteur est $f(t_n, u_n)$.

D'où l'équation (4.1) est remplacée par :

$$u_{n+1} = u_n + \int_{t_n}^{t_{n+1}} f\big(t, u(t)\big)dt \approx u_n + f(t_n, u_n)(t_{n+1} - t_n)$$

La méthode d'Adam-Bashforth d'ordre 2 est donnée par cette équation :

$$u_{n+1} = u_n + \frac{h}{2}(3f_n - f_{n-1})$$

La méthode d'Euler décrite ci-dessus est une méthode en une seule étape. C'est-à-dire, pour calculer l'approximation u_{n+1} de $u(t_{n+1})$, elle utilise seulement l'approximation de u_n. En revanche, le fonctionnement de la méthode multi étapes d'Adams-bashforth peut améliorer cette approximation parce que, lors de calcul de l'approximation de u_{n+1} d'autres approximations sont utilisées u_n et u_{n-1}.

Le réseau de Hopfield évolue selon une équation différentielle qui le caractérise. Pour la discrétisation de son équation différentielle, nous avons adopté la méthode d'Adams-bashforth.

- **Algorithme de système continu de Hopfield proposé pour le stable maximum**

Les adaptations de l'algorithme de système continu de Hopfield, présentées dans la littérature pour l'optimisation, ne traitent pas fondamentalement la bonne convergence de ce système[57]. Les éléments impliqués dans l'algorithme proposé sont:

Le vecteur u_i est l'entrée nette, v_i est la sortie et I_i est l'entrée externe pour le neurone i. Les W_{ij}, I_i et u_0 sont des valeurs constantes déterminées de manière appropriée par les caractéristiques du problème. Le h est une petite constante représentant un incrément de temps à chaque étape. Pour v_i est une fonction hyperbolique entre 0 et 1.

Le processus d'itération pour déterminer l'algorithme proposé :

- ***La phase d'initialisation***
    - Etape 0 :
        - Initialisation la matrice de poids w_{ij} et l'entré externe I_i
        - Initialisation des paramètres α, ε, φ et γ de la fonction énergétique
        - Initialisation la constante h par une petite valeur

- ***La phase d'itérations***

(Les étapes suivantes sont répétées jusqu'à ce que la fonction énergétique (E) cesse de diminuer).

- Etape 1 : Mettre à jour les neurones de réseau $u_{n+1} = u_n + \frac{h}{2}(3f_n - f_{n-1})$
- Etape 2 : Calculer de la sortie des neurones $v_n = \frac{1}{2}\left[1 + tanh\left(\frac{u_n}{u_0}\right)\right]$
- Etape 3 : Calculer la fonction énergétique
- Etape 4 : Comparaison entre la fonction énergétique actuelle $E(t)$ et précédente $E(t-1)$

  Si $E(t) > E(t-1)$ alors, nous sortons de l'algorithme.

  Sinon, nous devons continuer

- Etape 5 : calcule de f_{n+1} et revenons à l'étape 1.

Cet algorithme sera utilisé, par la suite, dans la résolution du problème de l'ensemble stable de taille maximale. Il représente l'évolution de réseau de Hopfield combiné avec une nouvelle approche d'Adams bashforth pour la discrétisation de son équation différentielle.

4.3.4 Résultats numériques et comparaisons

La présente section concerne la description de l'algorithme que nous proposons pour résoudre le problème de l'ensemble stable de taille maximale. Cet algorithme est testé sur des instances de la bibliothèque publique DIMACS[138].

Pour montrer l'intérêt pratique de l'approche ainsi proposée, plusieurs tests expérimentaux ont été effectués sur des instances de la bibliothèque publique DIMACS [138]. Sachant que les programmes ont été exécutés sur un ordinateur personnel i5 et RAM 4Go avec le langage Java. Les résultats sont présentés dans le tableau (4.3).

Il est commode de générer les sorties initiales de réseau de Hopfield aléatoirement comme suit : $x_i = 0.55 + 10^{-5}t$ où t est générée aléatoirement dans l'intervalle$[-0.5; 0.5]$.

Pour assurer la réalisabilité de point d'équilibre, nous choisissons les paramètres de réseau de Hopfield comme suit :$\alpha = 1.0250\,, \varepsilon = 10^{-6}\ et\ \gamma = 0.7$ et le paramètre ϕ a été calculé par l'équation $\phi = \alpha + \gamma + \varepsilon$.

graph	*V*	*|E|*	*α(G)*	*CHN Euler*	*CHN Adams*
				$\alpha_1(G)$	$\alpha_2(G)$
brock200_2	200	9876	12	11	11
brock200_4	200	13089	17	9	12
brock400_4	400	59765	33	6	9
brock800_2	800	208166	24	12	18
gen200_p0.9_44	200	17910	44	27	36
gen400_p0.9_55	400	71820	55	38	53
hamming8-4	256	20864	16	16	16
hamming10-4	1024	434176	40	40	40
keller4	171	9435	11	7	9
Keller5	776	225990	27	10	23
p_hat300_1	300	10933	8	8	8
p_hat300-3	300	33390	36	31	31
p_hat700-1	700	60999	11	11	11
p_hat700-2	700	121728	44	--	--
C125.9	125	6963	34	26	26
C250.9	250	27984	44	37	44
C1000.9	1000	450079	68	34	34
MANN_a27	378	70551	126	72	123

Tableau 4.3 Résultats de la simulation pour les instances de référence DIMACS

- -- : répétition des états instables.
- $\alpha(G)$: la taille optimale de l'ensemble stable
- $\alpha_1(G)$: la taille de l'ensemble stable obtenu par le système continu de Hopfield combiné par méthode d'Euler.
- $\alpha_2(G)$: la taille de l'ensemble stable obtenu par le système continu de Hopfield combiné par la méthode d'Adams-Bashforth.

Le tableau (4.3) montre que la méthode proposée permet d'aboutir à un bon ensemble stable de taille maximale des graphes de taille moyenne, notamment les graphes dont le nombre de sommets est inférieur ou égal à 400, ayant un nombre quelconque d'arcs. Il faut noter que ces résultats ont été obtenus en un temps raisonnable qui ne dépasse pas trois secondes pour la plupart des cas. La qualité de ces résultats peut être justifié par : la rapidité de réseau de Hopfield et de la bonne qualité des solutions obtenues (à titre d'exemple, la solution pour le graphe C250.9 est de taille d'un ensemble stable égal à 44 et dans un temps raisonnable).

4.4 Simulation du problème de satisfaction maximale des contraintes

Lorsqu'on modélise une application ou un problème réel sous la forme d'un problème de satisfaction de contrainte, il arrive souvent qu'il n'existe aucune solution ou il est extrêmement difficile de trouver cette solution. Le problème est alors qualifié de sur-contrainte. Dans cette situation, on cherche souvent à trouver un autre type de solution qui ne respecte pas toutes les contraintes, mais qui minimise le nombre de contraintes violées. Ce type de problème est connu dans le monde de la recherche scientifique par le problème de satisfaction maximale des contraintes. Dans ce cadre, on cherche à minimiser le nombre de contraintes violées par l'intermédiaire d'un critère d'optimisation. Alors, un problème MaxCSP est un problème de satisfaction de contraintes avec la recherche d'une solution qui minimise le nombre de contraintes violées[139].

La solution du problème MaxCSP est une attribution d'une valeur à chaque variable de son domaine tout en satisfaisant le maximum de contraintes. Dans ce contexte, nous avons proposé un nouveau modèle de programmation quadratique du problème MaxCSP qui consiste à minimiser une fonction quadratique sous des contraintes linéaires.

4.4.1 Modélisation du problème de satisfaction maximale des contraintes

Pour modéliser le problème de max-CSP, nous devons définir la variable de décision x_{ik} :

$$x_{ik} = \begin{cases} 1 \; si \; x_i = v_k \\ 0 \quad sinon \end{cases}$$

Avec $v_k \in D(x_i) \; \forall \; k \in \{1, \dots, d_i\} \; \forall i \in \{1, \dots, n\}$ et $d_i = |D(x_i)|$

Chaque contrainte $C(x_i, x_j)$ est associée à une relation R_{ij} tel que :

$$(v_r, v_s) \notin R_{ij} \Leftrightarrow x_{ir} x_{js} = 0$$

La fonction objectif est définie par :

$$f(x) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{r=1}^{d_i} \sum_{s=1}^{d_j} q_{irjs} x_{ir} x_{js}$$

Tel que q_{irjs} est représenté par :

$$q_{irjs} = \begin{cases} 1\ si\ (v_r, v_s) \notin R_{ij}, i \neq j \\ 0\ si\ (v_r, v_s) \in R_{ij}, i \neq j \\ 0\ si \qquad\qquad\quad i = j \end{cases}$$

Chaque variable x_i ne peut prendre qu'une seule valeur du domaine $D(x_i)$

$$\sum_{k=1}^{d_i} x_{ik} = 1\ \forall i \in \{1, \dots, n\}$$

Ces contraintes linéaires peuvent être présentées sous forme : $Ax = b$

$$A = \begin{pmatrix} 1 \cdots 1 \cdots 1 & 0 \cdots 0 \cdots 0 & \cdots & \cdots 0 \cdots 0 \cdots 0 \\ 0 \cdots 0 \cdots 0 & 1 \cdots 1 \cdots 1 & \cdots & \cdots 0 \cdots 0 \cdots 0 \\ 0 \cdots 0 \cdots 0 & 0 \cdots 0 \cdots 0 & \ddots & \vdots \quad\ \vdots \\ \vdots \quad\quad\ \vdots & \vdots \quad\quad\ \vdots & \vdots & \ddots \quad\ \vdots \\ 0 \cdots 0 \cdots 0 & 0 \cdots 0 \cdots 0 & 0 \cdots 0 \cdots 0 & \cdots 1 \cdots 1 \cdots 1 \end{pmatrix}, x = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1d_1} \\ x_{21} \\ \vdots \\ x_{2d_2} \\ \vdots \\ \vdots \\ x_{n1} \\ \vdots \\ x_{nd_n} \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Programmation quadratique en variables 0-1:

$$(QP) = \begin{cases} Min\ f(x) = \dfrac{1}{2} x^T Q x \\ sc \\ Ax = b \\ x \in \{0,1\}^N \end{cases}$$

Q : est une matrice carrée d'ordre N avec $N = \sum_{i=1}^{n} d_i$

b : est un vecteur de dimension n

A : est une matrice de $n \times N$

La modélisation se repose sur la modélisation du problème max-CSP avec une proposition d'une variable de décision S permettant la caractérisation d'une contrainte. Chaque contrainte C_{x_i,x_j} est associée à une variable de décision:

$$S_{ij} = \sum_{r=1}^{d_i} \sum_{s=1}^{d_j} q_{irjs} x_{ir} x_{js}$$

- **Théorème 1**

Soit x_{ik} la variable binaire définie par l'expression $\sum_{k=1}^{d_i} x_{ik} = 1$

Pour chaque contrainte C_{ij} entre deux variables x_i et x_j nous avons :

- *$S_{ij} = 1$ si et seulement si la contrainte C_{ij} est violée.*
- *$S_{ij} = 0$ si et seulement si la contrainte C_{ij} est satisfaite.*

La fonction objective $f(x)$ qui consiste à minimiser le nombre de contraintes violées est exprimée par la sommation suivante:

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} S_{ij}$$

- **Théorème 2**

Soit $V(QP)$ une valeur optimale du problème de programmation quadratique à variables bivalentes. Le nombre de contraintes violées du problème de satisfaction maximale de contraintes est égal à $V(QP)$.

Le modèle ainsi obtenu est le suivant :

$$(MaxCSP) \Leftrightarrow (QP) = \begin{cases} Min\ f(x) = \frac{1}{2} x^T Q x \\ sc \\ Ax = b \\ x \in \{0,1\}^N \end{cases}$$

- **Corollaire**

Le modèle proposé est capable d'identifier les contraintes qui sont violées lors de résolution du problème Max-CSP ainsi que le couple d'affectation (v_r, v_s) qui est responsable de cette violation. A vrai dire, si $S_{ij} = 1$, alors la contrainte qui est violée est C_{ij} et le couple d'affectation qui viole cette contrainte est (v_r, v_s).

- **Exemple**

On considère le problème de satisfaction des contraintes suivant (X, D, C, R):

$X = \{x_1, x_2, x_3\}$

$D(x_1) = \{v_1, v_2\}, D(x_2) = \{v_1, v_2, v_3\}$ *et* $D(x_3) = \{v_1, v_2\}$

$R_{12} = \{(v_1, v_2), (v_2, v_1), (v_2, v_3)\}$, $R_{13} = \{(v_1, v_1), (v_2, v_2)\}$

$R_{23} = \{(v_1, v_2), (v_2, v_1), (v_2, v_2), (v_3, v_2)\}$

Un couple (v_r, v_s) n'appartient pas à une relation est défini par $(v_r, v_s) \notin R_{ij} \Leftrightarrow x_{ir}x_{js} = 0$

L'ensemble des couples qui n'appartiennent pas à la relation :

$\{(v_1, v_1), (v_1, v_3), (v_2, v_2)\} \notin R_{12}$

$\{(v_1, v_2), (v_2, v_1)\} \notin R_{13}$

$\{(v_1, v_1), (v_3, v_1)\} \notin R_{23}$

La fonction objectif correspondante à cet exemple est :

$$f(x) = x_{11}x_{21} + x_{11}x_{23} + x_{12}x_{22} + x_{11}x_{32} + x_{12}x_{31} + x_{21}x_{31} + x_{23}x_{31}$$

La forme matricielle peut se rendre comme suit :

$$f(x) = \frac{1}{2}(x_{11} \quad x_{12} \quad x_{21} \quad x_{22} \quad x_{23} \quad x_{31} \quad x_{32}) \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{31} \\ x_{32} \end{pmatrix}$$

$$\sum_{k=1}^{d_i} x_{ik} = 1 \ \forall i \in \{1,2,3\} \Leftrightarrow \begin{cases} x_{11} + x_{12} = 1 \\ x_{21} + x_{22} + x_{23} = 1 \\ x_{31} + x_{32} = 1 \end{cases} \Leftrightarrow Ax = b$$

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \text{ et } b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Ce nouveau modèle d'optimisation met en interaction toutes les contraintes, et alors le problème de MaxCSP peut être traité globalement lors de la résolution. Par exemple, lors de la modélisation du problème MaxCSP, nous avons intégré toutes les contraintes binaires du problème MaxCSP dans notre modèle qui contient toutes les informations nécessaires sur le problème MaxCSP.

4.4.2 Description de l'algorithme proposé pour résoudre le problème Max-CSP

Afin de trouver une solution au problème de satisfaction maximale de contraintes via l'algorithme génétique, un algorithme a été développé. Rappelons qu'une solution est une affectation de valeurs à toutes les variables de sorte qu'un maximum de contraintes soient satisfaites. Le processus de résolution, par l'algorithme génétique, garantie une évolution des individus de la population par les mécanismes d'adaptation et sélection convenable. Alors, la chance de trouver une bonne solution augmente au fur et à mesure. Pour utiliser l'AG pour résoudre le problème de satisfaction maximale de contraintes, il devra définir les six éléments suivants :

- Le codage de solutions (individus) de la population,
- La génération de la population initiale,
- La fonction d'adaptation ou fitness,
- Le mécanisme de sélection,
- La diversification de la population,
- Le paramètre de dimensionnement.

Cette définition et représentation sont fondées sur des critères spécifiques du problème abordé. Cet algorithme commence par un ensemble de solutions (individus) appelé population. Ces solutions sont manipulées pour former une nouvelle population. Ceci est motivé par l'espoir, que la nouvelle population sera mieux que l'ancienne. Les solutions qui sont sélectionnées pour former la nouvelle population sont choisies en fonction de leur aptitude, le plus approprié, ils ont plus de chances d'être sélectionnées. Le processus de résolution de tout problème via l'algorithme génétique est défini de la façon suivante : Nous commençons par générer une population d'individus d'une manière aléatoire. Pour passer de la génération (i) à génération $(i+1)$, les deux opérations suivantes sont répétées pour tous les éléments de la population (i). L'opérateur de croisement est appliqué sur un couple de parents P_1 et P_2 sélectionné avec une probabilité pc (généralement autour de 0,8) pour produire un couple d'enfants C_1 et C_2. Un autre opérateur de mutation est appliqué pour un autre individu P_i selon une probabilité pm (pm est généralement plus petit que pc). Ce processus produit un nouvel enfant C_i. Le niveau d'adaptation des enfants (C_1, C_2) et C_i est ensuite évalué avant leur inclusion dans la nouvelle population.

L'idée originale de ce travail consiste à utiliser le modèle décrit dans ce chapitre et d'appliquer l'algorithme génétique afin de trouver une solution au problème de satisfaction maximale de contraintes dans un délai raisonnable. Dans ce contexte, l´étape la plus importante consiste à définir les éléments constituant l'algorithme génétique. Donc, le codage de solutions (individu), la génération de la population initiale, la fonction d'adaptation, l'opérateur de sélection et de diversification de la population sont définis à base des contraintes de notre modèle mathématique. A cet écart, ces éléments sont définis dans les sous sections suivantes.

- Codage d'individus de la population

Le codage est de type binaire pour produire des individus convenables à la solution qu'on cherche. En outre, la propriété importante est que chaque variable x_i doit prendre une valeur unique de son domaine $D(x_i)$. Ainsi, l'individu de la population doit être produit en respectant la contrainte linéaire. Enfin, chaque individu de notre population est codé de la manière suivante (Figure 4.10) :

d_1				d_2				d_i						d_n		
1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0

Figure 4.10 codage de la solution tout en respectant la contrainte

Lorsque le codage d'individu est défini, l'étape d'après consiste à générer la population initiale.

- La population initiale

La population initiale est générée aléatoirement. Rappelons que, le mécanisme de génération de la population initiale doit être capable de produire une population hétérogène qui servira comme une base pour les générations futures. Dans ce contexte, en se basant sur le type de codage, tous les gènes sont initialisés par "0" et pour chaque taille d_i dans chaque individu, nous générons aléatoirement une position dans le but d'insérer "1". Ainsi, la population initiale est générée aléatoirement de telle sorte que les contraintes linéaires sont satisfaites.

- Fonction d'adaptation

Afin d'évaluer les bonnes solutions et mettre en œuvre le mécanisme de sélection naturelle, la notion d'adaptation d'une solution est utilisée. Donc, notre objectif est de définir l'adaptation de chaque individu afin de mesurer la qualité de chaque individu. Cette fonction d'adaptation

est basée sur la valeur optimale $V(QP)$ du modèle. Pour définir cette fonction qui mesure la qualité de la solution représentée, nous calculons la valeur optimale OV_i de chaque individu et attribuer une valeur d'adaptation en fonction de cette valeur optimale. Dans ce contexte, un pourcentage d'adaptation d'un individu est calculé à l'aide de sa valeur optimale OV_i.

$$OV_i = NI_i^T QNI_i$$

L'idée principale est de calculer le pourcentage d'adaptation de l'individu. Ce pourcentage est calculé par la relation suivante :

$$FA_i = 100\frac{m - OV_i}{m}$$

Par exemple, si la valeur optimale d'un individu est supérieure strictement à zéro, alors cela montre une violation de contraintes.

- Mécanisme de sélection

Au niveau de la sélection, l'idée principale consiste à donner beaucoup de chance à l'individu le plus adapté d'être sélectionné et moins de chance pour le moins adapté. Pour cela, pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation à la résolution du problème. Afin de sélectionner un individu, on utilise le principe de la roue de la roulette qui consiste d'associer à chaque individu une partie dont la surface qui est proportionnelle à sa fitness. On reproduit ici le principe de tirage aléatoire utilisé dans les roulettes de casinos avec une structure linéaire. Ce type de sélection donne une bonne chance pour chaque individu bien adapté pour être sélectionné dans la génération future, et moins de chance à un individu moins adapté dans le but être éliminé dans la génération future (Figure 4.11).

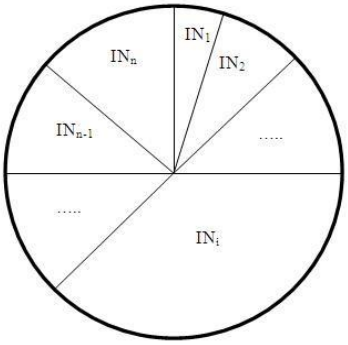

Figure 4.11 Roue de la roulette.

Lorsque le mécanisme de sélection est déterminé, il faut définir les opérateurs de diversification de la population pour bien exploré l'espace de solutions.

- Diversification de la population

La diversification de la population est une étape importante permettant l'exploration de l'espace de recherche. Cette diversification est garantie par les opérateurs de croisement et de mutation. Dans ce travail, un opérateur de croisement spécial, qui respecte les principales propriétés de notre modèle, est défini afin de garantir la production des enfants convenable, et l'opérateur de croisement ne s'applique pas dans n'importe quelle position dans l'individu, mais il doit être appliqué après chaque d_i dans l'individu (Figure 4.12 et 4.13).

	d_1							d_2							
P1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
P2	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 4.12 Parents P1 et P2

	d_1							d_2							
C1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 4.13 Fils C1 et C2 produits avec un point de croisement

L'opérateur de croisement recombine les gènes de deux individus existant dans la population. Cette recombinaison est appliquée dans une position spécifique qui s'appelle le point de croisement. Ce point est généré de façon aléatoire dans l'intervalle$[\,d_1, d_n]$. Noter que, la sélection des deux parents pour produire deux enfants est réalisée d'une manière aléatoire. L'opérateur de mutation vise à assurer l'exploration de l'espace de rechercher. Donc, pour appliquer cet opérateur, il faut respecter toujours les caractéristiques du codage d'individu. Dans cette situation, l'opérateur de mutation spéciale est appliqué. Cet opérateur consiste à changer d'un gène de valeur "1" par la valeur "0" et de choisir une autre d'une manière aléatoire en respectant la contrainte linéaire.

	d_1			d_2				d_3					d_4		
Pi	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0

	d_1			d_2				d_3					d_4		
Ci	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0

Figure 4.14 Operateur de mutation avec un point de mutation

Enfin, l'opérateur de mutation spécifique dans l'objectif de garantir l'exploration de l'espace de recherche en respectant la contrainte linéaire.

- Paramètres de dimensionnement

Après plusieurs essais, la taille de la population est fixée en fonction du nombre n de variables. Ainsi, la taille de la population P est définie par $P = 2n$. Le critère d'arrêt est défini afin de trouver une solution pour le problème de satisfaction maximale de contraintes. Le critère d'arrêt est le suivant : si un individu de la population dispose d'une fonction d'adaptation inferieur à un seuil, alors cet individu est une solution.

4.4.3 Résultats numériques

Afin de montrer l'intérêt pratique de l'approche proposée, des expériences sont réalisées pour résoudre certains problèmes typiques de différentes natures. Ces expériences sont effectuée dans un ordinateur personnel équipé d'un processeur 2.40 GHz et 4Go de RAM. La performance a été mesurée en termes de temps de calcul par seconde. Ce solveur est modélisé et implémenté par le langage Java. Le processus de résolution conduira à une affectation de valeurs à des variables telle que le nombre de contraintes satisfaites soit maximal. Par conséquent, le maximum des contraintes dans le problème (Max-CSP) soient satisfaites.
Une étude expérimentale a été représentée pour examiner la qualité de notre approche. Cette étude est basée sur le calcul de la performance des opérateurs. En fait, la qualité des solutions obtenues par notre approche a été évaluée en termes de rapport sur le rendement :

$$p = \frac{nombre\ de\ contraintes\ violées\ obtenue\ par\ notre\ approche}{meilleur\ nombre\ de\ contraintes\ violées\ obtenue\ par\ le\ meilleur\ solveur}$$

- **Ratio minimum** : le rapport entre le plus petit nombre de contraintes violées (meilleurs résultats obtenus par notre solveur) et le meilleur résultat existant dans la littérature obtenu par d'autres solveurs.
- **Le temps d'execution** : le temps consommé pour obtenir la solution à un certain nombre d'exécution.

Nous montrons que, pour chaque exemple, si le ratio minimum est inférieur à 1, alors l'approche proposée donne des meilleurs résultats que les autres, Autrement dit, donne une solution qui viole moins de contraintes en comparaison avec le meilleur solveur existant dans le benchmarks. Sinon, si le ratio minimum est supérieur à 1, alors l'approche proposée donne

une solution qui viole plus de contraintes par rapport au meilleur solveur existant dans la littérature, c'est à dire, n'arrive pas à trouver les résultats donnés par le meilleur solveur. Par ailleurs, si le ratio minimum est égal à 1, dans ce cas, les résultats donnés sont similaires que le meilleur solveur (tableau 4.4).

En comparaison avec d'autres solveurs de Max-CSP, le temps de résolution obtenu par notre approche est mieux que d'autres. En règle générale, notre solveur est très réussi, il arrive à trouver la solution aux problèmes de satisfaction maximale de contraintes en un minimum de temps que les autres solveurs de Max-CSP[140].

instance	NC	RB	Temps ben	RM	Temps app
maxcut-30-340-2	340	102	20.1629s	0.98	18.010s
maxcut-30-400-5	400	179	160.628s	0.94	123. 597s
maxcut-40-420-5	420	161	84.7451s	0.99	67.300s
maxcut-40-520-10	520	213	461.111s	0.98	398.02s
maxcut-50-560-10	560	212	1151.8s	1.03	790.326s
maxcut-50-580-10	580	221	1508.98s	1	681.250s
maxcut-60-420-2	420	135	176.564s	0.96	163.560s
maxcut-60-580-2	580	207	3316.9s	1.01	717.983s
vcsp-25-10-100-18-15	300	2	100.907s	1	116.846s
vcsp-15-10-100-93-14	105	73	12.3021s	1.09	11.587s
c-fat500-2	116111	474	22.0556s	1.1	13.592s
c-fat500-10	78623	374	14.0389s	1.05	12.042s

Tableau 4.4 Les instances de Max-CSP

Légende du tableau

- **NC** : nombre de contraintes
- **RB** : Résultat de benchmark
- **Temps ben** : temps d'exécution de benchmark
- **RM** : Ratio minimum
- **Temps app** : Temps d'exécution de l'approche proposée

Par exemple, pour les instances "maxcut-30-340-2", "maxcut-30-400-5", etc., le solveur proposé donne une solution qui viole moins de contraintes par rapport au meilleur solveur existant. Toutefois, dans certains cas, "maxcut-50-560-10", "vcsp-15-10-100-93-14", etc.,

notre approche viole ainsi des contraintes en plus d'autres solveurs. Enfin, nous pouvons conclure que les meilleurs résultats sont obtenus par cette approche.

4.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'optimisation des requêtes sur une base de données massives, à l'amélioration du réseau de Hopfield continu et aux problèmes de satisfaction maximale de contraintes.

La manipulation des données en base de données passe par des requêtes. Leur traitement nécessite beaucoup de temps et de ressources. Dans la première contribution, nous avons adopté une approche de matérialisation pour l'optimisation des requêtes dans les bases de données massives. Vue le coût de stockage qui n'a pas été étudié dans les différentes littératures existantes, nous avons modélisé le problème de sélection des vues sous forme d'un problème de satisfaction de contraintes pondérées tout en essayant de trouver le meilleur compromis entre tous les coûts de traitement des requêtes, de maintenance et de stockage. Cette modélisation a été à base d'un espace de recherche, ce dernier est construit par la fusion de huit requêtes afin d'avoir un espace avec un coût minimal. Nous avons évalué l'approche de matérialisation via l'algorithme génétique pour optimiser l'exécution des requêtes en utilisant le benchmark TCP-H[21]. Nous avons constaté que notre approche donne de meilleurs résultats pour notre cas d'expérimentation.

Dans la deuxième contribution, nous avons choisi de traiter un autre problème d'optimisation lié à la théorie de graphe. La méthode d'Adams bashforth et le réseau de Hopfield ont été la base de notre nouvelle approche de résolution du problème de l'ensemble stable de taille maximale d'un graphe. Nous avons testé cette approche sur des instances de la bibliothèque publique DIMACS[138]. Dans ce contexte, cette approche a montré une grande efficacité et rapidité. En plus, elle permet d'aboutir à la solution exacte pour les problèmes de l'ensemble stable de taille maximale de complexité moyenne.

Dans la troisième contribution, nous avons proposé une nouvelle approche pour résoudre les problèmes de satisfaction maximale de contraintes. Cette approche se compose de deux étapes intéressantes : La proposition d'un nouveau modèle du problème MaxCSP comme un problème de programmation quadratique sous des contraintes linéaires et l'utilisation de l'algorithme génétique pour résoudre ce dernier modèle. Il est également intéressant de noter que cette méthode peut être utilisée pour traiter les problèmes MaxCSP non-binaires après avoir converti ces dernier à des problèmes MaxCSP binaire[141]. Les résultats expérimentaux

montrent que notre méthode donne de bonnes solutions dans un temps minimal en comparaison avec d'autres solveurs.

Conclusion générale et perspectives

Les recherches réalisées dans le cadre de cette thèse de doctorat s'inscrivent dans la poursuite des recherches effectuées au sein de notre laboratoire de modélisation et calcul scientifique. Dans cet objectif, trois axes majeurs ont été développés.

Le premier axe de recherche a consisté à proposer une approche visant à optimiser les requêtes sur des données massives dans un environnement centralisé afin d'en augmenter la vitesse d'exécution. Il faut noter que, le temps de réponse de la requête étant le facteur le plus important pour évaluer la qualité de la prise de décision.

Comme nous l'avons vu dans cette étude, ce temps de réponse peut être influencé par plusieurs facteurs, principalement par la disponibilité des données. Au moment de soumettre une requête, il existe une différence importante entre le fait que cette requête possède des données pré-calculées ou non. La nécessité de réduire le temps de réponse des requêtes est satisfaite via des vues matérialisées. Cependant, cela nécessite une décision sur les vues qui doivent être matérialisés faisant ainsi du problème de sélection de vues la cible de plusieurs études dans la littérature. Le cycle d'exécution des requêtes s'étend de la capture des requêtes à la construction de l'espace de recherche, jusqu'à la matérialisation des vues. Les vues matérialisées permettent de stocker le résultat du traitement d'une requête sur un support physique dans le but de pouvoir l'utiliser plus tard. Ceci permet d'augmenter la vitesse d'exécution des requêtes. Un des défis majeurs consiste à sélectionner un ensemble optimal de vues. Cependant, dans les travaux de la littérature, la sélection de vues ne comprend pas tous les coûts. Dans ce travail de recherche, la sélection de vues a consisté à trouver le meilleur compromis entre tous les coûts : les coûts de traitement, coûts de maintenance et les coûts de stockage.

Nous avons proposé une nouvelle approche visant à optimiser la performance des requêtes sur des données massives dans un environnement centralisé afin d'augmenter leurs vitesses d'exécution. Nous avons décrit les démarches permettant la construction d'un espace de recherche optimal. Cette approche a consisté à modéliser le problème de sélection de vues sous forme d'un problème de satisfaction de contraintes pondérées. Nous avons utilisé le benchmark TCP-H pour construire notre espace de recherche en fusionnant huit requêtes afin d'avoir un environnement avec un coût minimal. Cet environnement nous a permis de sélectionner les meilleures vues à matérialiser tout en considérant tous les coûts de traitement de requêtes, de maintenance et de stockage via l'algorithme génétique.

Le second axe de recherche a visé à étudier le problème du stable maximum issu de la théorie des graphes. Nous définissons le problème consistant à trouver un ensemble indépendant ou stable comme un ensemble de sommets dans un graphe tel qu'aucun d'entre eux n'est adjacent à un autre.

L'approche proposée ici a consisté à utiliser le système continu de Hopfield combiné à la méthode d'Adams-bashforth. Pour mettre en œuvre ce système continu, nous avons modélisé le problème du stable maximum sous une forme d'un problème d'optimisation quadratique. Par la suite, nous avons représenté ce problème comme une fonction énergétique. La discrétisation de système continu de Hopfield a permis d'améliorer sa performance. La convergence de l'approche proposée a été mesurée. La validation de notre approche sur le problème du stable maximum a montré son efficacité en comparaison avec le réseau de Hofield classique. En effet, notre approche a prouvé sa performance en trouvant des stables de tailles maximales.

Dans le troisième axe, nous avons proposé une nouvelle approche pour résoudre les problèmes de satisfaction maximale de contraintes. Cette approche se compose de deux étapes intéressantes : La proposition d'un nouveau modèle du problème MaxCSP comme un problème de programmation quadratique sous des contraintes linéaires et l'utilisation de l'algorithme génétique pour résoudre ce dernier modèle. Il est également intéressant de noter que cette méthode peut être utilisée pour traiter les problèmes MaxCSP non-binaires après avoir convertir ces dernier à des problèmes MaxCSP binaire. Les résultats expérimentaux montrent que notre méthode donne de bonnes solutions dans un temps minimal en comparaison avec d'autres solveurs.

L'une des principales perspectives de recherche qui apparaissent à l'issue de cette thèse est la modélisation du problème de sélection de vues sous forme de réseaux de neurones de Hopfield. Ainsi, il serait intéressant d'utiliser le système continu de Hopfield amélioré comme une technique de résolution de ce modèle. Une autre perspective aussi importante est liée à l'extraction, la transformation et le chargement des données connu dans la littérature sous le nom d'ETL (extract load transform). En effet, cet outil pouvant encore être perfectionné, reste un sujet ouvert dans le monde informatique. Nous envisageons d'étudier la possibilité d'améliorer l'optimisation des requêtes ETL sur tout le cycle de traitement des données, c'est-à-dire de l'étape d'acquisition des données à la dernière étape de prise de décision en passant par la matérialisation.

References

[1] J. Zhang, Y. Liu, and H. Wang, “Image Preprocessing Methods Used in Meteorological Measurement of the Temperature Testing System,” *J. Geosci. Environ. Prot.*, vol. 4, no. 11, pp. 1–5, 2016.

[2] W. H. Inmon, *Building the Data Warehouse*. John Wiley & Sons, Inc. New York, NY, USA ©1992, 1992.

[3] H. Gupta and I. S. Mumick, “Selection of views to materialize in a data warehouse,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 1, pp. 24–43, 2005.

[4] J. Yang, K. Karlapalem, and Q. Li, “A Framework for Designing Materialized Views in Data Warehousing Environment,” *Icdcs’97*, pp. 458–465, 1997.

[5] S.-J. Lim and Y.-K. Ng, “A Hybrid Fragmentation Approach for Distributed Deductive Database Systems,” *Knowl. Inf. Syst.*, vol. 3, no. 2, pp. 198–224, 2001.

[6] I. Mami and Z. Bellahsene, “A survey of view selection methods,” *ACM SIGMOD Rec.*, vol. 41, no. 1, p. 20, 2012.

[7] M. El Alaoui, K. El moutaouakil, and M. Ettaouil, “Weighted Constraint Satisfaction and Genetic Algorithm To Solve the View Selection Problem,” *Int. J. Database Manag. Syst. (IJDMS)*, vol. 9, no. 4, pp. 11–20, 2017.

[8] M. El Alaoui, K. El Moutaouakil, and M. Ettaouil, “A multi-step method to calculate the equilibrium point of the Continuous Hopfield Networks: Application to the max-stable problem,” *WSEAS Trans. Syst. Control*, vol. 12, pp. 418–425, 2017.

[9] V. Harinarayan, “Implementing data cubes efficiently,” *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pp. 205–216, 1996.

[10] J. O. Chan, “Optimizing Data Warehousing Strategies,” *Commun. IIMA*, vol. 5, no. 1, pp. 1–14, 2005.

[11] C. Loboz, S. Smyl, and S. Nath, “DataGarage: Warehousing Massive Performance Data on Commodity Servers,” *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 1447–1458, 2010.

[12] S. S. Conn, “OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis,” *Proceedings. IEEE SoutheastCon, 2005.*, no. May 2005, pp. 515–520, 2005.

[13] N. Jukic, “Online Analytical Processing (OLAP) for Decision Support,” *Handb. Decis. Support Syst. 1*, pp. 1–24, 2008.

[14] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, “Efficient OLAP operations in spatial data warehouses,” *Spat. Temporal Databases*, vol. 2121, pp. 443–459, 2001.

[15] S. Chaudhuri and U. Dayal, “An overview of data warehousing and OLAP technology,” *ACM SIGMOD Rec.*, vol. 26, no. 1, pp. 65–74, 1997.

[16] E. Weippl, O. Mangisengi, W. Essmayr, F. Lichtenberger, and W. Winiwarter, “An Authorization Model for Data Warehouses and OLAP,” *Proc. IEEE Work. Secur. Distrib. Data Warehous. 2001*, no. FEBRUARY 2002, 2001.

[17] J. Zhao, “Designing Distributed Data Warehouses and OLAP Systems,” *Inf. Syst.*, pp. 254–263, 2014.

[18] F. Teklitz, “The Simplification of Data Warehouse Design,” p. 20, 2000.

[19] P. Vassiliadis, C. Quix, Y. Vassiliou, and M. Jarke, “Data warehouse process management,” *Inf. Syst. J.*, vol. 26, no. 3, pp. 205–236, 2001.

[20] B. P. Rahmadi Wijaya, “An Overview and Implementation of Process in Data Warehouse,” pp. 70–74, 2015.

[21] P. O. Neil, B. O. Neil, and X. Chen, “Star Schema Benchmark - Revision 3,” *Tech. rep.*, 2009.

[22] L. BELLATRECHE, “Techniques d ’optimisation des requetes dans les data warehouses,” *6th Int. Symp. Program. Syst. (ISPS 03), Alger, Alger. 2003.*, p. 81–98., 2003.

[23] G. Bukhbinder, M. Krumenaker, and A. Phillips, “Insurance Industry Decision Support: Data Marts, OLAP, and Predictive Analytics,” *Casualty Actuar. Soc. Forum*, pp. 171–197, 2005.

[24] S. Sharma and R. Jain, “Modeling ETL process for data warehouse: An exploratory study,” *Int. Conf. Adv. Comput. Commun. Technol. ACCT*, pp. 271–276, 2014.

[25] R. Goswami, D. K. Bhattacharyya, M. Dutta, and J. K. Kalita, “Approaches and issues in view selection for materialising in data warehouse,” *Int. J. Bus. Inf. Syst.*, vol. 21, no. 1, pp. 17–47, 2016.

[26] D. Theodoratos and T. Sellis, “Dynamic Data Warehouse Design,” *DaWaK*, pp. 1–10, 1999.

[27] I. Mami and Z. Bellahsene, “A survey of view selection methods,” *ACM SIGMOD Rec.*, vol. 41, no. 1, p. 20, 2012.

[28] T. V. V. Kumar and S. Kumar, “Materialised view selection using differential evolution,” *Int. J. Innov. Comput. Appl.*, vol. 6, no. 2, pp. 102–113, 2014.

[29] J. X. Yu, X. Yao, C. H. Choi, and G. Gou, “Materialized View Selection as Constrained Evolutionary Optimization,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 33, no. 4, pp. 458–467, 2003.

[30] G. Gou, J. X. Yu, and H. Lu, “A* search: An efficient and flexible approach to materialized view selection,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 36, no. 3, pp. 411–425, 2006.

[31] D. Yang, M. Huang, and M. Hung, “Efficient Utilization of Materialized Views in a Data Warehouse,” *PAKDD 2002 Adv. Knowl. Discov. Data Min.*, pp. 393–404, 2002.

[32] C. A. Dhote and M. S. Ali, “Materialized view selection in data warehousing: A survey,” *Journal of Applied Sciences*, vol. 9, no. 3. pp. 401–414, 2009.

[33] W.-Y. Lin and I.-C. Kuo, “A Genetic Selection Algorithm for OLAP Data Cubes,” *Knowl. Inf. Syst.*, vol. 6, no. 1, pp. 83–102, 2004.

[34] M. Lawrence, “Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses,” *Proc. 8th Annu. Conf. Genet. Evol. Comput. - GECCO ’06*, p. 699, 2006.

[35] N. A. R. Yousri, K. M. Ahmed, and N. M. El-Makky, “Algorithms for selecting materialized views in a data warehouse,” *3rd ACS/IEEE Int. Conf. Comput. Syst. Appl. 2005*, vol. 2005, pp. 89–96, 2005.

[36] C. Zhang, X. Yao, and J. Yang, “Evolving Materialized Vieews in Data Warehouse,” *1999 {C}ongress {E}volutionary {C}omputation*, pp. 823–829, 1999.

[37] A. Gupta and I. S. Mumick, “Maintenance of Materialized Views: Problems, Techniques, and Applications,” *IEEE Data Eng. Bull.*, vol. 18, pp. 3–18, 1995.

[38] R. Derakhshan and F. Dehne, “Simulated Annealing for Materialized View Selection in Data Warehousing Environment.,” *24th IASTED Int. Conf. Database Appl.*, pp. 89–94, 2006.

[39] H. Kong, J. Phuboon-ob, and R. Auapanwiriyakul, “Analysis and comparison of algorithm for selecting materialized views in a data warehousing environment,” pp. 682–686, 2006.

[40] A. Kerkad, L. Bellatreche, and D. Geniet, “La Fragmentation Horizontale Revisitée: Prise en Compte de l’Interaction de Requêtes,” pp. 117–132, 2013.

[41] K. Aouiche, P.-E. Jouve, and J. Darmont, “Clustering-Based Materialized View Selection in Data Warehouses,” *Lect. Notes Comput. Sci. Incl. Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinforma.*, vol. 4152 LNCS, no. 1, pp. 81–95, 2007.

[42] B. Suchyukorn and R. Auepanwiriyakul, “Re-Optimization MVPP Using Common Subexpression for Materialized View Selection,” vol. 7, no. 7, pp. 1114–1121, 2013.

[43] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, “Optimizing queries with materialized views,” *Proc. Elev. Int. Conf. Data Eng.*, pp. 190–200, 1995.

[44] M. Lee, I. Science, and F. Gainesville, “Speeding Up Warehouse Physical Design Using A Randomized Algorithm,” *Management*, vol. 1999, no. April, pp. 1–9, 1999.

[45] Z. A. Talebi, R. Chirkova, and Y. Fathi, “Exact and inexact methods for solving the problem of view selection for aggregate queries,” *Int. J. Bus. Intell. Data Min.*, vol. 4, no. 3/4, p. 391, 2009.

[46] B. Ashadevi and R. Balasubramanian, “Cost Effective Approach for Materialized Views Selection in Data Warehousing Environment,” *Ijcsns*, vol. 8, no. 10, p. 236, 2008.

[47] J. N. Li, Z. A. Talebi, R. Chirkova, and Y. Fathi, “A formal model for the problem of view selection for aggregate queries,” *Adv. Databases Inf. Syst.*, vol. 3631, no. 321635, pp. 125–138, 2005.

[48] A. Bauer and W. Lehner, “On Solving the View Selection Problem in Distributed Data Warehouse Architectures 3 . Problem Space of the Distributed View,” *Management*, 2003.

[49] I. Mami, R. Coletta, and Z. Bellahsene, “Modeling view selection as a constraint satisfaction problem,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6861 LNCS, no. PART 2, pp. 396–410, 2011.

[50] M. Lawrence and A. Rau-Chaplin, “Dynamic view selection for OLAP,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4081 LNCS, pp. 33–44, 2006.

[51] B. Shah, K. Ramachandran, and V. Raghavan, “A Hybrid Approach for Data Warehouse View Selection 1,” pp. 1–56, 2015.

[52] K. Haddouch, M. Ettaouil, and C. Loqman, “Continuous hopfield network and quadratic programming for solving the binary constraint satisfaction problems,” *J. Theor. Appl. Inf. Technol.*, vol. 56, no. 3, pp. 362–372, 2013.

[53] M. Mitchell, “Genetic algorithms: An overview,” *Complexity*, vol. 1, no. 1, pp. 31–39, 1995.

[54] C. Van Der Malsburg, “Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms,” *Brain Theory Proc. First Trieste Meet. Brain*

Theory, Oct. 1--4, 1984, no. 1, pp. 245–248, 1986.

[55] H. J. M. Peters, "A perceptron," pp. 1–5, 1988.

[56] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proc. Natl. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[57] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, 1985.

[58] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proc. Natl. Acad. Sci.*, vol. 81, no. 10, pp. 3088–3092, 1984.

[59] A. Krenker, J. Bešter, and A. Kos, "Introduction to the Artificial Neural Networks," *Eur. J. Gastroenterol. Hepatol.*, vol. 19, no. 12, pp. 1046–1054, 2011.

[60] C. Darwin, "On the origins of species by means of natural selection," *London: Murray*, p. 247, 1859.

[61] K. Sastry, D. Goldberg, and G. Kendall, "Genetic Algorithms," *Compute*, pp. 97–125, 2005.

[62] J. H. Holland, "Genetic Algorithms - Computer programs that 'evolve' in ways that resemble natural selection can solve complex problems even their creators do not fully understand," *Scientific American*. pp. 66–72, 1992.

[63] C. Reeves, "Genetic algorithms," *Handb. metaheuristics*, pp. 109–140, 2010.

[64] K. Szeto and J. Zhang, "Adaptive genetic algorithm and quasi-parallel genetic algorithm: Application to knapsack problem," *Large-Scale Sci. Comput.*, pp. 189–196, 2006.

[65] R. L. Y. M. A. Rangel-Merino, J. L. López-Bonilla, "Optimization Method based on Genetic Algorithms," *Apeiron, Vol. 12, No. 4*, vol. 12, no. 4, pp. 393–408, 2005.

[66] M. Vali, "Rotational Mutation Genetic Algorithm on optimization Problems," pp. 1–10, 2013.

[67] S. Ding, C. Su, and J. Yu, "An optimizing BP neural network algorithm based on genetic algorithm," *Artif. Intell. Rev.*, vol. 36, no. 2, pp. 153–162, 2011.

[68] P. Diaz-Gomez and D. Hougen, "Initial Population for Genetic Algorithms: A Metric Approach.," *Proc. 2007 Int. Conf. Genet. Evol. Methods*, pp. 43–49, 2007.

[69] E. Cantú-Paz, "A Survey of Parallel Genetic Algorithms," *Calc. Paralleles Reseaux Syst. Repart.*, vol. 10, no. 2, pp. 141–171, 1998.

[70] Rudolph G., "Convergence properties of canonical genetic algorithms.," *IEEE Trans.*

Neural Networks, vol. 1, no. 5, pp. 96–101, 1994.

[71] D. Thierens and D. E. Goldberg, "Convergence models of genetic algorithm selection schemes," *Lect. Notes Comput. Sci.*, vol. 866, no. 90, pp. 119–129, 1994.

[72] W. Fan, E. A. Fox, P. Pathak, and H. Wu, "The effects of fitness functions on genetic programming-based ranking discovery for web search," *J. Am. Soc. Inf. Sci. Technol.*, vol. 55, no. 7, pp. 628–636, 2004.

[73] T. Blickle and L. Thiele, "A Comparison of Selection Schemes used in Genetic Algorithms," *TIK-Report*, vol. 2, no. 11, pp. 311–347, 1995.

[74] R. Malhotra, N. Singh, and Y. Singh, "Genetic Algorithms : Concepts , Design for Optimization of Process Controllers," *Comput. Inf. Sci.*, vol. 4, no. 2, pp. 39–54, 2011.

[75] M. Noraini and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," *World Congr. Eng.*, vol. II, no. 978-988-19251-4–5, pp. 4–9, 2011.

[76] R. Hinterding, H. Gielewski, and T. Peachey, "The Nature of Mutation in Genetic Algorithms.," *Proc. Sixth Int. Conf. Genet. Algorithms*, pp. 65–72, 1995.

[77] N. L. Law and K. Y. Szeto, "Adaptive genetic algorithm with mutation and crossover matrices," *IJCAI Int. Jt. Conf. Artif. Intell.*, no. 1, pp. 2330–2333, 2007.

[78] F. Manya and C. Gomes, "Solution Techniques for Constraint Satisfaction Problems," *Intel. Artif.*, vol. 7, no. 19, pp. 243–267, 2003.

[79] U. Montanari, "Network of Constraints: Fundamental Properties and Applications to Picture Processing," *Inf. Sci.*, vol. 7, pp. 97–132, 1974.

[80] P. Van Beek, "Backtracking Search Algorithms," pp. 85–134, 2006.

[81] N. Sadeh and M. S. Fox, "Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem," *Artif. Intell.*, vol. 86, no. 1, pp. 1–41, 1996.

[82] C. Lecoutre, F. Boussemart, and F. Hemery, "Au coeur de la consistance d'arc," *Actes de JNPC'03*, pp. 233–247, 2003.

[83] J. Larrosa and T. Schiex, "Solving weighted CSP by maintaining arc consistency," *Artif. Intell.*, vol. 159, no. 1–2, pp. 1–26, 2004.

[84] J. Larrosa, "Node and arc consistency in weighted CSP," *18th Natl. Conf. Artif. Intell. (AAAI-02), 14th Innov. Appl. Artif. Intell. Conf. (IAAI-02), Jul 28-Aug 1 2002*, vol. 1, no. Mackworth, pp. 48–53, 2002.

[85] M. Charikar, K. Makarychev, and Y. Makarychev, "Near-optimal algorithms for maximum constraint satisfaction problems," vol. V, pp. 62–68, 2007.

[86] F. Rossi, K. B. Venable, and T. Walsh, “Preferences in Constraint Satisfaction and Optimization,” *AI Mag.*, vol. 29, no. 4, p. 58, 2009.

[87] D. S. Johnson, “A Brief History of NP-Completeness , 1954 – 2012,” vol. I, pp. 359–376, 2012.

[88] N. Mamoulis and K. Stergiou, “Solving Non-binary CSPs Using the Hidden Variable Encoding,” *Proc. 7th Int. Conf. Princ. Pract. Constraint Program. ({CP} 2001)*, pp. 168–182, 2001.

[89] S. C. Brailsford, C. N. Potts, and B. M. Smith, “Constraint satisfaction problems: Algorithms and applications,” *Eur. J. Oper. Res.*, vol. 119, no. 3, pp. 557–581, 1999.

[90] A. Gascon and R. C. Leachman, “A Dynamic Programming Solution to the Dynamic, Single-Machine Scheduling Problem,” *Oper. Res.*, vol. 36, no. 1, pp. 50–56, 1988.

[91] I. P. Gent, C. Jefferson, and P. Nightingale, “Complexity of n-Queens Completion,” *J. Artif. Intell. Res.*, vol. 59, pp. 815–848, 2017.

[92] S. Khan, M. Bilal, M. Sharif, M. Sajid, and R. Baig, “Solution of n-Queen problem using ACO,” *INMIC 2009 - 2009 IEEE 13th Int. Multitopic Conf.*, 2009.

[93] L. Kroc, A. Sabharwal, and B. Selman, “Counting solution clusters in graph coloring problems using belief propagation,” *22nd NIPS*, pp. 873–880, 2008.

[94] A. Hertz, M. Plumettaz, and N. Zufferey, “Variable space search for graph coloring,” *Discret. Appl. Math.*, vol. 156, no. 13, pp. 2551–2560, 2008.

[95] C. Jaillet, “Solving the Langford problem in parallel.”

[96] I. Miguel and Q. Shen, “Solution Techniques for Constraint Satisfaction Problems: Foundations,” *Artif. Intell. Rev.*, vol. 15, no. 4, pp. 243–267, 2001.

[97] W. Pang and S. D. Goodwin, “A graph based backtracking algorithm for solving general CSPs,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2671, pp. 114–128, 2003.

[98] R. Smith, C. Estan, and S. Jha, “Backtracking algorithmic complexity attacks against a NIDS,” *Proc. - Annu. Comput. Secur. Appl. Conf. ACSAC*, pp. 89–98, 2006.

[99] S. N. Ndiaye and C. Terrioux, “A generic bounded backtracking framework for solving CSPs.”

[100] N. Sadeh, K. Sycara, and Y. Xiong, “Backtracking techniques for the job shop scheduling constraint satisfaction problem,” *Artif. Intell.*, vol. 76, no. 1–2, pp. 455–480, 1995.

[101] E. C. Chi, K. Lange, and L. Angeles, “Techniques for Solving Sudoku Puzzles,” *Hum.*

Genet., vol. 7088, p. 16, 2012.

[102] D. Frost and R. Dechter, "Looking at Full Looking Ahead," *Proc. Second Int. Conf. Princ. Pract. Constraint Program.*, pp. 539–540, 1996.

[103] C. Likitvivatanavong, Y. Zhang, E. C. Freuder, and J. C. Bowen, "Maintaining arc consistency using adaptive domain ordering," *IJCAI Int. Jt. Conf. Artif. Intell.*, pp. 1527–1528, 2005.

[104] C. Bessiere, "Arc-consistency for non-binary dynamic CSPs," *ECAI 92. 10th Eur. Conf. Artif. Intell. Proceedings, 3-7 Aug. 1992*, pp. 23–27, 1992.

[105] C. Bliek and D. Sam-Haroud, "Path consistency on triangulated constraint graphs," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 1, pp. 456–461, 1999.

[106] C. Bessière, "Arc-consistency and arc-consistency again," *Artif. Intell.*, vol. 65, no. 1, pp. 179–190, 1994.

[107] F. Bacchus and A. Grove, "On the forward checking algorithm," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 976, pp. 292–309, 1995.

[108] C. Bessière, P. Meseguer, E. C. Freuder, and J. Larrosa, "On forward checking for non-binary constraint satisfaction," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1713, pp. 88–102, 1999.

[109] W. Faber, N. Leone, and F. Ricca, "A backjumping technique for disjunctive logic programming," *CEUR Workshop Proc.*, vol. 142, pp. 216–230, 2005.

[110] S. Esmeir and S. Markovitch, "Lookahead-based algorithms for anytime induction of decision trees," *Twenty-first Int. Conf. Mach. Learn. - ICML '04*, p. 33, 2004.

[111] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artif. Intell.*, vol. 14, no. 3, pp. 263–313, 1980.

[112] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus, "Conflict ordering search for scheduling problems," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9255, pp. 140–148, 2015.

[113] I. Gent, K. Stergiou, and T. Walsh, "Decomposable constraints," *Artif. Intell.*, vol. 123, no. 1–2, pp. 133–156, 2000.

[114] M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen, "Structural Decomposition Methods and What They are Good For," *Leibniz Int. Proc. Informatics*, pp. 12–28, 2011.

[115] J. Clausen, "Branch and bound algorithms-principles and examples," *Dep. Comput.*

Sci. Univ. …, pp. 1–30, 1999.

[116] H. A. Eiselt and C.-L. Sandblom, “Branch and Bound Methods,” *Integer Program. Netw. Model.*, pp. 205–228, 2000.

[117] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex, “Solving Max-SAT as Weighted CSP.,” *Proc. Ninth Int. Conf. Princ. Pract. Constraint Program.*, pp. 363–376, 2003.

[118] R. J. Wallace, “Directed Arc Consistency Preprocessing as a Strategy for Maximal Constraint Satisfaction,” *Springer, Berlin, Heidelb.*, vol. 923, pp. 1–17, 2005.

[119] J. Larrosa and P. Meseguer, “Partition-Based Lower Bound for Max-CSP,” no. i, pp. 303–315, 1999.

[120] M. C. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner, “Soft arc consistency revisited,” *Artif. Intell.*, vol. 174, no. 7–8, pp. 449–478, 2010.

[121] F. Baumann, S. Berckey, and C. Buchheim, “Exact Algorithms for Combinatorial Optimization Problems with Submodular Objective Functions,” *Facet. Comb. Optim.*, pp. 271–294, 2013.

[122] G. Verfaillie and T. Schiex, *Constraint Satisfaction Problems*. 2013.

[123] P. Bajpai and M. Kumar, “Genetic algorithm–an approach to solve global optimization problems,” *Indian J. Comput. Sci. Eng.*, vol. 1, no. 3, pp. 199–206, 2010.

[124] K. Stefanidis and E. Pitoura, “Preference Modeling and Orders,” *Database*, no. 1, pp. 1–7.

[125] J. Larrosa, E. Morancho, and D. Niso, “On the Practical use of Variable Elimination in Constraint Optimization Problems: ‘Still-life’ as a Case Study,” *J. Artif. Intell. Res.*, vol. 23, pp. 421–440, 2005.

[126] F. Fioretto, E. Pontelli, W. Yeoh, and R. Dechter, “Accelerating exact and approximate inference for (distributed) discrete optimization with GPUs,” *Constraints*, pp. 1–43, 2017.

[127] M. C. Silaghi, “A suite of secure multi-party computation algorithms for solving distributed constraint satisfaction and optimization problems,” 2004.

[128] J. Yang, K. Karlapalem, and Q. Li, “Algorithms for materialized view design in data warehousing environment,” *Vldb*, no. 1, pp. 136–145, 1997.

[129] J.-H. Yang and C.-J. Chung, “ASVMRT: Materialized View Selection Algorithm in Data Warehouse,” *J. Inf. Process. Syst.*, vol. 2, no. 2, pp. 67–75, 2006.

[130] W. Xu, D. Theodoratos, and C. Zuzarte, “Computing closest common subexpressions for view selection problems,” *Proc. 9th ACM Int. Work. Data Warehous. Ol. - Dol.*

’06, p. 75, 2006.

[131] J. E. Gallardo, C. Cotta, and A. J. Fernández, “Solving Weighted Constraint Satisfaction Problems with Memetic/Exact Hybrid Algorithms,” *J. Artif. Intell. Res.*, vol. 35, pp. 533–555, 2009.

[132] S. De Givry and G. Katsirelos, “Clique Cuts in Weighted Constraint Satisfaction.”

[133] M. Farber, “Characterizations of strongly chordal graphs,” *Discrete Math.*, vol. 43, no. 2–3, pp. 173–189, 1983.

[134] P. S. Kumar and C. E. V. Madhavan, “Minimal vertex separators of chordal graphs,” *Discret. Appl. Math.*, vol. 89, no. 1–3, pp. 155–168, 1998.

[135] D. Kumlander, “A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search,” *Proc. 5th Int’l Conf. Model. Comput. …*, 2004.

[136] M. Xiao and H. Nagamochi, “Exact Algorithms for Maximum Independent Set,” no. 1, pp. 1–35, 2013.

[137] N. S. Dattani, “Linear Multistep Numerical Methods for Ordinary Differential Equations,” *Univ. Waterloo*, pp. 1–10, 2008.

[138] D. G. B. I. and B. U. Bounds, “DIMACS,” 1992. [Online]. Available: http://www.info.univ-angers.fr/pub/porumbel/graphs/.

[139] A. Mackworth, “Consistency in a Network of Relations,” *Artif. Intell.*, vol. 8, no. 1, pp. 99–118, 1977.

[140] “MAX-CSP 2008 Competition,” 2008. [Online]. Available: http://www.cril.univ-artois.fr/CPAI08/results/.

[141] H. Bennaceur, C. Lecoutre, and O. Roussel, “A Decomposition Technique for Max-CSP,” *Ecai 2008, Proc.*, vol. 178, p. 500+, 2008.

Annexe

Requête Q1

```
SELECT N_NAME, MIN(PS_SUPPLYCOST)
FROM PART, PARTSUPP, SUPPLIER, NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY
AND S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

Requête Q2

```
SELECT N_NAME, SUM(L_QUANTITY)
FROM ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY N_NAME;
```

Requête Q3

```
SELECT COUNT(PS_SUPPKEY)
FROM PARTSUPP, PART
WHERE P_PARTKEY = PS_PARTKEY
AND P_BRAND <> 'BRAND#45'
AND NOT P_TYPE LIKE '%BRASS%'
AND P_SIZE IN (9, 19, 49)
GROUP BY P_NAME;
```

Requête Q4

```
SELECT S_NAME, SUM(PS_SUPPLYCOST)
FROM SUPPLIER, PARTSUPP, PART
WHERE S_SUPPKEY = PS_SUPPKEY
AND P_PARTKEY = PS_PARTKEY
AND P_BRAND <> 'BRAND#45'
AND NOT P_TYPE LIKE '%BRASS%'
AND P_SIZE IN (9,19,49)
GROUP BY S_NAME;
```

Requête Q5

```
SELECT C_MKTSEGMENT, SUM(L_DISCOUNT)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY C_MKTSEGMENT;
```

Requête Q6

```
SELECT P_BRAND, O_ORDERPRIORITY, MAX(L_TAX)
FROM PART, LINEITEM, ORDERS
WHERE P_PARTKEY = L_PARTKEY
AND L_ORDERKEY=O_ORDERKEY
AND L_SHIPMODE='FOB'
GROUP BY P_BRAND, O_ORDERPRIORITY;
```

Requête Q7

```
SELECT N_NAME, SUM(L_QUANTITY)
FROM ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
AND L_SHIPMODE='FOB'
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY N_NAME;
```

Requête Q8

```
SELECT N_NAME, AVG(L_DISCOUNT)
FROM REGION, NATION, SUPPLIER, PARTSUPP, PART, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND PS_PARTKEY = P_PARTKEY
AND P_BRAND <> 'BRAND#45'
AND NOT P_TYPE LIKE '%BRASS%'
AND P_SIZE IN (9,19,49)
AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```