Université de Montréal

Affectation de composantes basée sur des contraintes énergétiques dans une architecture multiprocesseurs en trois dimensions

par
Martin Deldicque

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

juin, 2014

RÉSUMÉ

La lithographie et la loi de Moore [43] ont permis des avancées extraordinaires dans la fabrication des circuits intégrés. De nos jours, plusieurs systèmes très complexes peuvent être embarqués sur la même puce électronique [20].

Les contraintes de développement de ces systèmes sont tellement grandes qu'une bonne planification dès le début de leur cycle de développement est incontournable. Ainsi, la planification de la gestion énergétique au début du cycle de développement est devenue une phase importante dans la conception de ces systèmes.

Pendant plusieurs années, l'idée était de réduire la consommation énergétique en ajoutant un mécanisme physique une fois le circuit créé, comme par exemple un dissipateur de chaleur. La stratégie actuelle est d'intégrer les contraintes énergétiques dès les premières phases de la conception des circuits. Il est donc essentiel de bien connaître la dissipation d'énergie avant l'intégration des composantes dans une architecture d'un système multiprocesseurs de façon à ce que chaque composante puisse fonctionner efficacement dans les limites de ses contraintes thermiques. Lorsqu'une composante fonctionne, elle consomme de l'énergie électrique qui est transformée en dégagement de chaleur.

Le but de ce mémoire est de trouver une affectation efficace des composantes dans une architecture de multiprocesseurs en trois dimensions en tenant compte des limites des facteurs thermiques de ce système.

Mots clés : Recherche opérationnelle, Affectation quadratique, Recherche avec tabous, Architecture de multiprocesseurs, Dégagement de chaleur, Positionnement d'éléments, Choix des bus.

ABSTRACT

Lithography and Moore's law [43] have led to extraordinary advances in integrated circuits manufacturing. Nowadays, many complex systems can be embedded on the same chip [20].

Development constraints of these systems are so significant that a good planning from the beginning of the development stage is essential. Thus, the planning of energy management at the beginning of the development cycle has become important in the design of these systems.

For several years, the idea was to reduce energy consumption by adding a cooling system once the circuit is created, a heat sink for example. The current strategy is to integrate energy constraints in the early stages of circuits design. It is therefore important to know the energy dissipation before the integration of the components in the architecture of a multiprocessor system so that each component can work within the limits of its thermal stresses. When a component is running, it consumes electric energy which is converted into heat.

The aim of this thesis is to find an efficient assignment of components in a multiprocessor system architecture in three dimensions, taking into account the limits of its thermal factors.

Key words: Operations research, Quadratic assignment, Taboo search, Architecture of multiprocessors, Heat dissipation, Assignment of elements, Choosing bus.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, Jacques A. Ferland, qui a été très présent et accessible dans toutes les étapes de l'élaboration de ce mémoire. Ses précieux conseils en recherche opérationnelle m'ont aidé à me diriger dans la bonne voie, et les nombreux échanges réguliers que nous avons eus m'ont permis de m'ajuster constamment et donc de gagner un temps précieux. Je remercie aussi mon co-directeur, Étienne Élie, pour son aide précieuse dans le domaine électronique, et son point de vue pratique sur les applications industrielles relatives au sujet traité dans ce mémoire.

Je remercie aussi ma famille et mes amis, qui m'ont donné du courage tout au long de mon parcours et qui ont participé à la relecture de ce mémoire.

CHAPITRE 1

INTRODUCTION

1.1 Contexte

Dans un contexte industriel où la performance joue un rôle de plus en plus important, nous ne pouvons plus nous passer de systèmes de calculs multiprocesseurs. Leur développement au cours de la dernière décennie a été très rapide, et de nouveaux problèmes physiques ont alors vu le jour. L'affectation des composantes dans une architecture multiprocesseurs est devenue un défi à l'heure actuelle.

Le but est de placer les composantes (processeurs et mémoires) dans une structure en trois dimensions, de façon à optimiser une certaine fonction de coût. Plusieurs méthodes existent dans la littérature pour résoudre ce genre de problème, que ce soit en utilisant des résultats de théorie de graphes [38], ou encore certaines heuristiques. Malheureusement, dans la plupart des cas, ces méthodes ne tiennent pas compte de la capacité des liens entre les différentes composantes. En travaillant avec un système en trois dimensions, il est nécessaire de reconsidérer le processus d'affectation, notamment pour garantir de bons résultats au niveau énergétique.

En effet, le fonctionnement d'une composante entraîne une consommation d'énergie électrique et donc un dégagement de chaleur qu'il faut contrôler pour éviter une surchauffe, qui engendrerait un dysfonctionnement majeur - voire total - du système. Dans ce mémoire, nous travaillons sur un système en trois dimensions, car à l'heure actuelle, la miniaturisation est essentielle pour garder les produits attractifs. Une structure en trois dimensions permet donc de relier plus facilement les composantes, en économisant de l'espace. L'étude de la dissipation d'énergie avant l'intégration des composantes permet alors à chaque composante de fonctionner efficacement dans les limites de ses contraintes thermiques.

1.2 Problématique

Le but de ce mémoire est d'affecter les composantes (processeurs et mémoires) sur une puce, de façon à minimiser un certain coût, qui prend en compte la performance et la qualité thermique du système.

La performance tient compte de la distance qui sépare deux composantes, mais aussi de l'interaction entre ces deux composantes. Plus les composantes qui interagissent sont proches, plus la performance est élevée.

La qualité thermique du système repose aussi sur l'agencement des composantes, mais cette fois, nous allons regarder la position des composantes dans la structure en trois dimensions. Par exemple, deux composantes qui interagissent beaucoup et qui sont au centre de la structure vont produire de la chaleur qu'il va être difficile de dissiper, cela représente donc un problème thermique qui sera pris en compte dans l'évaluation du coût.

1.3 Contribution

L'approche que nous avons utilisée pour aborder la problématique a été de décomposer le problème en deux parties. Dans la première, nous affectons les éléments (processeurs et mémoires) aux nœuds de l'architecture. Dans la seconde, nous choisissons les bus à affecter aux liens de l'architecture en fonction des accès entre les composantes. Ces deux sous-problèmes sont modélisés à l'aide de modèles de programmation mathématique en nombres entiers, et leurs résolutions font appel à des méthodes métaheuristiques.

Dans sa thèse [20], Élie propose de résoudre le problème d'affectation des bus en utilisant une méthode par perturbations. Il y explique que la théorie de la perturbation est une méthode de résolution par approximation utilisée en mathématiques appliquées pour trouver des solutions aux problèmes les plus complexes. L'idée de base de la résolution est de trouver une solution exacte à un problème proche de celui que l'on veut résoudre et de changer cette solution en ajoutant ou soustrayant un terme aux variables du problème pour obtenir la solution désirée. Il s'agit d'une méthode largement utilisée en physique

et chimie [22].

Cette méthode par perturbations n'est cependant pas adaptée aux problèmes de grande taille, or la réduction de la taille des transistors et la complexité fonctionnelle des circuits intégrés vont exiger de résoudre dans le futur des problèmes qui font intervenir plusieurs centaines de composantes. C'est pourquoi nous proposons de résoudre ce problème en utilisant une méthode heuristique. En utilisant une hypothèse de radialité, nous formerons un problème de programmation mathématique que nous résoudrons à l'aide d'un algorithme de recherche avec tabous, en l'adaptant aux besoins fonctionnels du problème.

1.4 Structure du mémoire

Ce mémoire est organisé comme suit. Le chapitre 2 présente une revue de littérature de l'architecture des systèmes sur puce électronique. Dans le chapitre 3, nous allons décrire le problème d'affectation quadratique et présenter deux méthodes de résolution classiques. Ensuite, dans le chapitre 4, nous allons voir comment affecter les processeurs et les mémoires dans une structure en trois dimensions. Nous avons notamment développé une méthode qui permet de déduire une matrice de flot à partir des interactions entre les groupes de processeurs et les familles de mémoires. D'autre part, nous verrons comment matérialiser la prise en compte du dégagement de chaleur dans le processus d'affectation et d'optimisation de la structure.

Nous traiterons le problème de la sélection des bus dans le chapitre 5, où nous ferons l'analogie entre notre problème et un problème d'affectation dans un réseau électrique. Pour résoudre ce problème, Élie utilise une méthode de perturbation qui consiste à modifier légèrement une solution pour en obtenir une meilleure en procédant avec un mécanisme pas à pas. Dans ce mémoire, nous utilisons une méthode heuristique pour résoudre ce problème d'optimisation.

Le chapitre 6 traitera des résultats obtenus et d'une discussion sur les méthodes utilisées. Nous conclurons enfin en rappelant les points clés de ce mémoire et en proposant d'autres approches de résolution.

CHAPITRE 2

REVUE DE LITTÉRATURE

Dans ce chapitre, nous allons présenter un état de l'art de la conception de circuits intégrés multiprocesseurs sur puce électronique, puis nous verrons quelles techniques permettent de prendre en compte le dégagement de chaleur lors de l'affectation des composantes aux nœuds d'une architecture en trois dimensions.

2.1 Conception de circuits intégrés

L'industrie de la microélectronique fait face à de nouveaux défis depuis quelques années : la complexité des circuits intégrés et la miniaturisation font de la conception de circuits intégrés une problématique d'actualité. Les systèmes sur puce se composent habituellement de trois éléments : les processeurs, les mémoires et les liens de communication.

Les processeurs sont d'une grande diversité et le choix du modèle adéquat dépend des besoins de l'architecture. Dans sa thèse, Elie explique que les processeurs doivent être « flexibles, modulables et paramétriques, afin de pouvoir les intégrer facilement dans une architecture. L'approche la plus efficace est le processeur dédié (Application Specification Integrated Circuit [ASIC] ou Digital Signal Processor [DSP]). Les processeurs dédiés sont construits sur un jeu d'instructions très réduit qui ne traite que des besoins spécifiques pour des domaines d'application bien précis [32]. Toutefois, ils peuvent servir souvent à des classes d'application comme le multimédia ou le traitement d'images. Parmi ces processeurs, nous avons le TriMedia TM32 [18], [29] qui est un processeur pour les applications multimédia qui permet de transférer des données parallèles et de manière continuelle. Un autre exemple de processeur dédié est le Micro Engine ME du système IXP2850 de réseau WAN et LAN, qui est utilisé pour exécuter des fonctions à filins d'exécution multiples sur les paquets. Vu le nombre important de paquets dans un processeur de réseau, plusieurs ME sont utilisés pour permettre un traitement pa-

rallèle. Plusieurs autres processeurs dédiés comme le SPE [24] sont proposés par des entreprises afin de traiter un problème bien particulier dans l'architecture des systèmes On-Chip Multiprocessor (OCM).

Bien qu'efficaces, les processeurs dédiés ne sont pas les plus populaires dans la conception des OCM ; le plus grand problème des ASIC ou des DSP, est qu'ils coûtent très cher en ressources et en temps à développer, ce qui augmente considérablement le coût total du système. Le processeur embarqué ARM (Advanced RISC Machine) [5], est le plus utilisé par les fabricants des OCM. L'avantage du ARM par rapport aux processeurs dédiés cités ci-dessus, est qu'il est générique et n'est pas physiquement fabriqué par ses fournisseurs. Ainsi, chaque processeur est conçu sur un noyau ARM au-dessus duquel des spécifications propres aux clients sont ajoutées. Nous citons comme exemple l'architecture du processeur de ST Nomadik qui contient le processeur ARM926E-JS [6] qui est basé sur le noyau ARM9.

Un autre noyau de processeur souvent utilisé dans la conception des OCM est le MIPS (Microprocessor without Interlocked Pipeline Stages). Comme le processeur ARM, le MIPS est un noyau de processeur qui permet de créer plusieurs processeurs dérivés. Bien qu'utilisé dans les ordinateurs et dans les serveurs de Silicon Graphic, on le retrouve souvent comme processeur embarqué dans des OCM. Ainsi, dans l'architecture du Nexperia [18], une variante de MIPS appelée le PR3940 a été développée pour s'occuper du système d'exploitation et pour contrôler les exécutions de certaines tâches.

Nous reconnaissons que les processeurs cités ci-dessus ne constituent pas une liste exhaustive et que plusieurs autres processeurs embarqués ont été développés et sont abondamment documentés dans la littérature.

Les processeurs softcores sont proposés par des fabricants de Field-Programmable Gate Array (FPGA). Ces processeurs sont spécifiquement dédiés et synthétisables uniquement sur les circuits programmables de leurs fabricants. C'est le cas du Nios de Altera [3] et du MicroBlaze de Xilinx [66]. Plusieurs applications ont utilisé les processeurs MicroBlaze de Xilinx comme dans le cas de [53], [44] et [31], et des processeurs Nios pour le prototypage OCM à moindre coût. Quelques travaux sont à souligner dans le cas de l'intégration de Nios dans une application OCM comme dans les références

[50], [68] et [19].

En ce qui concerne les processeurs généraux (CPU), dont les architectures les plus populaires sont basées sur le 80x86, leur conception permet de les utiliser pour la résolution de problèmes plus généraux. Ils sont souvent moins chers parce qu'ils sont faits pour une utilisation générale et une production de masse. Cependant, leurs tailles sont généralement importantes et ils sont souvent lents pour être intégrés dans des systèmes de haute performance. De nos jours, seule le PowerPC, qui est en réalité une dérivée du MIPS, a été embarqué sur le FPGA Virtex-5 FX70T de Xilinx [65], et est utilisé pour les traitements d'ordre général.

En conclusion, nous pouvons dire qu'il y a une diversité de processeurs qui peuvent être utilisés pour la conception des OCM et que le choix ne dépend que des besoins de l'architecture ; ce qui fait qu'ils ne constituent pas un problème de performance.

Contrairement aux processeurs, le choix des mémoires embarquées est limité, vu le nombre réel de sortes de mémoires disponibles dans l'industrie. Toutefois, leur organisation, leur hiérarchie et les protocoles de routages sont importants pour la conception de systèmes performants, et sont souvent différents d'un système à l'autre. Nous retrouvons par exemple le cas du TRIPS [10] ou du Tilera Tile64 [63] dont les mémoires sont organisées en deux niveaux de cache L1 et L2, et utilisent un contrôleur matériel alors que le Teraflops de Intel (80 processeurs) et le IBM Cyclops-64 utilisent des contrôleurs logiciels [14] et [28].

L'un des problèmes que les mémoires embarquées rencontrent est leur taille. Elles doivent être assez petites à cause de l'espace disponible et assez large pour favoriser le facteur de localité entre les données et les fonctions. Pour résoudre ce problème, des solutions comme la conception de mémoire sans cache ou l'utilisation des mémoires statiques à la place des mémoires dynamiques (plus complexe) [40] sont proposées. Ainsi, pour réduire la taille des processeurs, la consommation d'énergie et réduire la chaleur, les auteurs de l'article [8] ont proposé une alternative à la mémoire cache appelée Scratch-pad Memory, qui est une circuiterie composée d'un décodeur et d'une table mémoire. Cette circuiterie occupe une partie de l'espace d'adressage de la mémoire principale et

crée une association entre le reste de l'espace mémoire.

Dans [67], Xu et al. ont fait une étude de différentes organisations hiérarchiques des mémoires dans un système OCM. Ils ont démontré premièrement que l'utilisation d'une hiérarchie de la mémoire cache L1 et L2 dans les processeurs peut augmenter la performance de calcul d'une application dont la fréquence d'accès à la mémoire est très élevée. Cependant, la différence entre la bande passante d'une mémoire externe au OCM et sa mémoire interne peut entrainer une sérieuse perte de performance lorsque plusieurs filins sont actifs. Deuxièmement, ils ont montré que les congestions dans le OCM sont causées par des accès importants des processeurs aux mémoires. Ainsi, il est nécessaire de calculer adéquatement le chemin de données qui connecte chaque processeur du système à la mémoire principale. Ils pensent par ailleurs qu'il n'est pas adéquat de privilégier une optimisation de la communication entre les composantes du système sur la puce au dépend de certaines paramètres comme la chaleur et la consommation de l'énergie. En théorie, il est démontré que chaque application a un modèle de communication unique [7]. Ainsi, Dally et Towles pensent toutefois que la performance d'un système sur puce dépend de son accès aux données, de la hiérarchie des mémoires et surtout du réseau d'interconnexion sur la puce [13].

Le problème d'accès aux mémoires est malheureusement le goulot d'étranglement. La recherche d'équilibre entre la hiérarchie de la mémoire, le chemin des données, l'accès à la mémoire externe et la recherche de chemins critiques est devenue un problème d'optimisation combinatoire dont la complexité dépend de l'application traitée.

Les liens entre les unités sont constitués d'une interconnexion ou d'un câblage de fils parallèles [52], [1] qui permettent de distribuer l'horloge, le courant électrique (Vcc, GND, Reset), la terre et d'autres signaux comme les données (DATA) et les commandes (CMD) entre les processeurs et les mémoires. Le besoin d'une transmission rapide des données à travers le système et d'une qualité des données et de services transmis exigent comme dans le cas des processeurs et des mémoires que les liens soient efficaces et performants. En effet, avec la diminution de la taille des transistors, les liens de communications se sont rapprochés et de nouveaux problèmes se sont posés. Parmi ces problèmes,

nous avons la non fiabilité des données, les bruits électriques et les interférences. Par ailleurs, les spécifications physiques des lignes utilisées jusqu'à présent dans l'industrie de semi-conducteurs limitent grandement leurs vitesses de transmission.

Deux catégories de liens de communications sont utilisées dans la conception des OCM. Les liens parallèles et les liens sériels. Les liens les plus utilisés sont les liens parallèles car ils permettent naturellement de transmettre des mots de données à chaque cycle d'horloge. Aussi, ils contribuent à la dissipation de la chaleur mais posent d'autres problèmes comme l'espace occupée, les bruits électriques, les erreurs dues à l'électro migration et à l'interférence. D'autre part, les liens sériels deviennent de plus en plus populaires parce qu'ils permettent de corriger certains problèmes comme la réduction de l'espace utilisé, l'intégrité des données et la diminution des erreurs, mais en posent d'autres comme la chaleur et les délais supplémentaires [15]. Nous remarquons ainsi que dans les OCM, les liens constituent les maillons faibles de la performance. Pour corriger ce problème, plusieurs travaux ont été proposés.

Dans [59], Tamhankar et al. ont proposé une approche agressive appelé Terror qui tolère les fautes temporelles causées par les aléas physiques afin d'augmenter la performance de la communication. La méthodologie réduit le nombre de mémoires tampons entre les noeuds NoC et propose un protocole qui consiste à corriger les erreurs par Go-Back N présenté dans la référence [62]. D'autres travaux se sont concentrés sur la correction d'erreur pour augmenter la performance des liens de communications entre les transmetteurs de données (e.g., [54]).

Pour corriger les problèmes que posent les transferts de données parallèles, Kangmin et al. ont présenté un outil qui propose un transfert en série [35] basé sur une méthode d'encodage de la donnée entre les noeuds. Leur but étant de minimiser le nombre de transitions sur une ligne de transmission sérielle en utilisant les corrélations entre les mots des données successives. Cette corrélation minimise le nombre de '1' logiques et augmente le nombre de '0'. Ce qui réduit l'énergie consommée et la chaleur dissipée.

Une autre approche pour optimiser l'utilisation des liens de communication sur les puces est d'utiliser des communications asynchrones. Dans [60], Teifel et Manohar ont présenté une architecture matérielle des émetteurs et des récepteurs de données sans

horloge, ceci afin d'alléger les noeuds des générateurs d'horloges qui sont considérés comme coûteux. Le mode de transmission des données dans leur solution est sériel. Toutefois, des multiplexeurs du côté des émetteurs et des démultiplexeurs du côté des récepteurs sont utilisés pour assembler et désassembler les données parallèles qui viennent des noeuds. Contrairement aux transmetteurs de données qui utilisent plusieurs cycles d'horloge pour assembler ou désassembler les données, leur méthode utilise un lien à trois fils avec une architecture basée sur une machine à états finis (Finite-State Machine [FSM]) à trois états et sur un protocole d'anneaux à jeton. Aussi, l'architecture proposée permet-elle de faire une synchronisation dynamique entre le taux des bits envoyés par l'émetteur et celui des bits échantillonnés par le récepteur. Signalons que dans la littérature, plusieurs autres travaux qui traitent des transmissions asynchrones dans les NoC ont été proposés, e.g. [16].

Plusieurs autres travaux ont été proposés pour la conception de liens de communication performants comme [60], [41], [15], [34], [49] et [54]. Toutefois, nous n'avons pas trouvé dans la littérature une approche qui propose une conception de liens de communication dédiés. Nous pensons que pour chaque système à concevoir, des liens dédiés entre les noeuds selon la particularité de chaque arc peuvent augmenter la performance du système tout entier. »

2.2 Affectation dans les systèmes sur puce

La phase d'affectation des composantes sur une puce est primordiale lors de la création d'un circuit intégré. En effet, cette affectation doit respecter un ensemble de contraintes d'optimisation. Comme l'indique Elie dans sa thèse, « l'affectation comme la partition matérielle est faite après la synthèse en trois étapes : l'affectation globale qui permet de définir grosso modo quelles espaces les composantes occupent sur la puce, la légalisation qui permet de résoudre les problèmes de chevauchement des composantes et des portes logiques sont réglés, et enfin, le raffinement qui permet de détailler l'affectation globale. Durant ces dernières années, plusieurs travaux ont été consacrés aux problèmes d'affectations dans la conception des circuits intégrés. Cependant, des études

récentes ont montré qu'avec la diminution de la taille des transistors et l'accroissement de la densité fonctionnelle des circuits intégrés, les approches conventionnelles d'affectation ne produisent plus les résultats optimaux. La plupart des méthodes et outils d'affectation basés sur des architectures de bus [39], [27], ont une limitation inhérente pour traiter la complexité des circuits intégrés actuels. Des travaux sont proposés dans l'industrie et par des unités de recherche académiques pour corriger ces problèmes [12], [33], [9]. Ces travaux exploitent les nouvelles architectures et topologies comme les OCM. Traditionnellement, deux tendances sont à considérer dans les méthodes d'affectation. Il y a les méthodes qui préconisent la séparation de l'affectation du Hardware/Software Codesign (HSC). Dans tous les cas, l'objectif est d'optimiser des paramètres comme la distribution de l'horloge, minimiser la consommation d'énergie ou maximiser la dissipation de la chaleur dans les circuits. Parmi les approches utilisées pour le placement (avec partition ou pas), nous avons la méthode de la coupe minimum [33], [48], les méthodes analytiques [51], [61] qui sont considérées comme les meilleures en terme d'optimalité, et les méthodes hybrides qui combinent les deux approches précédentes [55]. L'utilisation d'algorithmes méta heuristiques qui permettent de faire des explorations dans de très larges espaces de recherche ont permis aux méthodes analytiques de régler les problèmes de densités fonctionnelles des OCM. Nous constatons malheureusement que les problèmes d'affection sont pour la plupart des méthodes souvent a posteriori à la synthèse et qu'à cette étape de la conception, certaines limitations des phases de développement précédant la synthèse et jumelées à la partition HSC peuvent entrainer des pertes de performance difficiles à corriger. »

2.3 Prise en compte des contraintes thermiques

La complexité des circuits actuels induit de nouvelle contraintes lors de leur conception, et notamment au niveau thermique. En effet, la diminution de la taille des transistors ou encore la structure en trois dimensions des nouvelles puces font que la prise en compte du dégagement de chaleur est un paramètre à prendre en compte lors de l'analyse de la performance d'un système. Comme indiqué par Élie, beaucoup d'énergie est dé-

pensée dans le stockage et les transferts de données sur une puce. La littérature propose quelques solutions pour améliorer le transfert de données, notamment en travaillant sur l'optimisation du design des mémoires tampon ([30]). Quelques études plus récentes se concentrent sur la conception de circuits qui réduisent la consommation électrique des commutateurs *crossbar* tout en maximisant l'interconnexion des composantes, comme l'explique Élie.

La nécessité de réduire les transferts de données dans le but de réduire les dégagements de chaleur donne lieu à la résolution d'un problème difficile qui ajoute un niveau de complexité à l'affectation des composantes. Dans sa thèse, Élie utilise une méthode par perturbations pour affecter les bus sur l'architecture. Il s'agit d'une méthode « pas à pas » utilisée notamment dans le domaine de la chimie en perturbant légèrement une solution réalisable pour en obtenir une meilleure.

Dans ce mémoire, nous utiliserons une méthode heuristique pour résoudre le problème de la sélection des bus au chapitre 5. En effet, en faisant une hypothèse de configuration radiale, nous utiliserons une recherche avec tabous pour minimiser une fonction de coût faisant intervenir un terme lié au dégagement de chaleur.

CHAPITRE 3

PROBLÈME D'AFFECTATION QUADRATIQUE

3.1 Définition du problème d'affectation quadratique

Le problème d'affectation quadratique (QAP) est un problème d'optimisation combinatoire qui a été introduit par Koopmans et Beckmann [37]. Il s'agit d'un problème NP-difficile qui est considéré comme étant l'un des plus difficiles à résoudre de façon exacte.

Dans un problème d'affectation quadratique, N unités doivent être assignées à N différents lieux, en sachant que la distance entre les lieux i et j est d_{ij} et qu'un flot ayant pour valeur f_{rs} va de l'unité r à l'unité s. Le problème se formule donc comme suit :

$$\bar{z} = \min_{\phi} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} f_{\phi(i)\phi(j)} \qquad \phi : \text{permutation de } \{1, \cdots, N\}$$

où $\phi(i)$ désigne l'unité affectée au lieu i. Il s'agit donc de minimiser le produit *distance* x *flot*.

Nous comptons de nombreuses applications du problème d'affectation quadratique, comme par exemple la distribution de services médicaux dans un hôpital [21], ou encore le problème du voyageur de commerce [47]. Pour d'autres exemples, le lecteur est invité à consulter le travail de Finke *et al.* [23] qui donne un aperçu plus complet.

Des solutions optimales sont généralement trouvées pour des instances avec $N \leq 36$. Pour des problèmes de plus grande taille, nous utilisons des heuristiques qui permettent de donner de bonnes solutions pour des problèmes de taille $N \approx 100$ ou moins. Cependant il est à noter qu'un problème de voyageur de commerce avec près de 25000 villes en Suède a été résolu de façon exacte [4].

3.2 Techniques de résolution

Nous distinguons plusieurs méthodes heuristiques pour résoudre le problème d'affectation quadratique. Nous allons présenter ici les deux méthodes qui seront utilisées dans la suite.

3.2.1 Colonies de fourmis

L'idée d'imiter le comportement des fourmis pour obtenir de bonnes solutions pour des problèmes d'optimisation combinatoire a été soulevée par Dorigo, Maniezzo et Colorni [17]. Le principe de cette méthode est basé sur la façon dont les fourmis procèdent pour aller chercher de la nourriture en dehors de la fourmilière, puis retourner à bon port. Au début, les fourmis empruntent des chemins aléatoires. Par la suite, dès qu'une fourmis a trouvé de la nourriture, elle retourne à la fourmilière avec une certaine quantité de nourriture.

Pendant ce trajet retour, la fourmi laisse sur son passage une phéromone chimique. Les phéromones s'évaporent au cours du temps, c'est-à-dire que la trace laissée par une fourmi diminue si d'autres fourmis n'empruntent pas ce chemin. Cette trace chimique permet aux autres fourmis d'être guidées vers la source de nourriture, et la quantité de phéromone dépend de la quantité de nourriture trouvée. Après un certain temps, le chemin de la fourmilière à la source de nourriture est indiqué par une forte concentration de phéromones, et plus le nombre de fourmis qui trouvent de la nourriture est grand, plus la trace de phéromones est importante.

Comme les sources de nourriture les plus proches de la fourmilière sont atteintes plus fréquemment que celles qui sont plus éloignées, les traces de phéromones qui indiquent une source de nourriture proche s'intensifient plus rapidement. Les phéromones sont donc un moyen pour les fourmis de trouver un chemin vers une source de nourriture, et revenir ensuite à la fourmilière.

La transposition de ce principe en un algorithme repose sur l'analogie faite entre :

1. Le domaine de recherche des vraies fourmis et l'ensemble des solutions réalisables

du problème combinatoire

2. La quantité de nourriture dans une source et la fonction objectif

3. La trace des phéromones et l'implémentation d'une mémoire adaptative

Très schématiquement, en supposant que p fourmis sont disponibles, l'algorithme se présente ainsi [57] :

```
1) Initialiser les traces.
2) Répéter en parallèle pour chacune des p fourmis et
tant qu'un critère d'arrêt n'est pas satisfait :
  2a) Construire une nouvelle solution à l'aide des
      informations contenues dans les traces et une
      fonction d'évaluation partielle.
  2b) Évaluer la qualité de la solution.
  2c) Mettre à jour les traces.
```

L'analogie entre la trace des phéromones et l'implémentation d'une mémoire adaptative est la plus difficile à réaliser. En effet, la création et la mise à jour des traces dépend du problème que nous voulons résoudre. Taillard propose une implémentation qui donne d'excellents résultats [56] en y ajoutant notamment une recherche locale qui se résume à une méthode de descente.

En partant d'une solution initiale, nous modifions légèrement la structure de cette solution pour en obtenir une meilleure, jusqu'à ne plus pouvoir obtenir d'améliorations. Pour modifier une solution, nous définissons un ensemble de mouvements qui peuvent être appliqués à toute solution.

Nous avons choisi cet algorithme de colonies de fourmis implémenté par Taillard pour résoudre le problème d'affectation quadratique.

3.2.2 Recherche avec tabous

La recherche avec tabous a été proposée par Glover [26] à la fin des années 1980. L'idée de base est de définir un voisinage (comme dans la plupart des méthodes heuris-

tiques) qui donne lieu à un ensemble de mouvements qui peuvent être appliqués à une solution donnée, pour en produire une nouvelle dans son voisinage.

Parmi les solutions qui appartiennent au voisinage de la solution courante, nous choisissons celle qui améliore le plus la fonction objectif. S'il n'y en a pas, cela veut probablement dire que nous avons atteint un optimum local, auquel cas nous choisissons celle qui dégrade le moins la solution courante. Pour éviter de retourner à cet optimum local, il faut interdire le mouvement inverse. C'est là qu'intervient la liste de tabous. Cette liste contient un nombre s d'éléments qui caractérisent chacun un mouvement interdit. C'est donc cette liste qui empêche de revenir en arrière pendant un certain nombre d'itérations, en interdisant certains mouvements. Cependant, il est possible que la liste de tabous contienne un ou plusieurs mouvements qui seraient intéressants, comme par exemple des mouvements qui amélioreraient la meilleure solution connue. Pour parer ces éventualités, un critère d'aspiration est défini. Ce dernier permet d'accepter des mouvements tabous s'ils sont jugés intéressants.

C'est cette technique de résolution que nous avons choisi d'implémenter pour traiter le problème de la sélection des bus que nous verrons dans le chapitre 5. Nous nous sommes basés sur les améliorations apportées par Taillard [58], qui fournissent une méthode facile à implémenter, avec peu de paramètres, et qui pour autant donne de très bons résultats. De plus, nous utiliserons une liste de tabous de taille variable, pour parer le phénomène de cycles qui sera détaillé à la section 5.3.3.

CHAPITRE 4

AFFECTATION DES COMPOSANTES

L'objectif de ce mémoire est d'affecter des composantes à des places prédéfinies dans une architecture en trois dimensions pour en faire un système efficace. Ces composantes sont des processeurs et des mémoires dont les interactions sont fournies par une matrice d'incidence C qui constitue une donnée de base. Chaque élément c_{ij} de cette matrice est un entier qui correspond au nombre de fois que le processeur i accède à la mémoire j. Puisqu'en général le nombre de sommets de l'architecture est beaucoup plus petit que le nombre de processeurs et de mémoires, notre approche consiste d'abord à générer des regroupements (cellules) contenant des processeurs et des mémoires ayant une interaction importante. Pour ce faire, nous modélisons le processus à l'aide d'un problème de formation de cellules dont nous cherchons à maximiser l'autonomie. Ceci a fait l'objet du mémoire de maîtrise de M.Z. Ahadri [2], et nous résumons son approche dans la section 4.1.

En résolvant le problème de formation de cellules, nous obtenons une matrice d'incidence permutée où nous pouvons identifier les K cellules générées. Cette matrice permutée est ensuite utilisée pour former une matrice d'interaction IA entre les groupes de processeurs et les familles de mémoires. Cette matrice d'interaction sera finalement utilisée pour générer une matrice de flot F, qui constituera une entrée du problème d'affectation quadratique.

La deuxième entrée nécessaire à la résolution du problème (QAP) est une matrice de distance D. Pour générer cette matrice associée au graphe $G = (V,E)$ de l'architecture, constitué d'un ensemble de nœuds V et d'un ensemble d'arêtes E, nous utilisons l'algorithme de Floyd-Warshall.

À ce stade, nous avons alors en main la matrice de distance et la matrice de flot que le problème d'affectation quadratique utilise pour affecter chaque groupe de processeurs et chaque famille de mémoires sur le réseau de l'architecture en trois dimensions, dans le

but de minimiser la somme des produits *distance* x *flot*. Nous allons détailler dans cette section ces différentes étapes que nous résumons à la Figure 4.1.

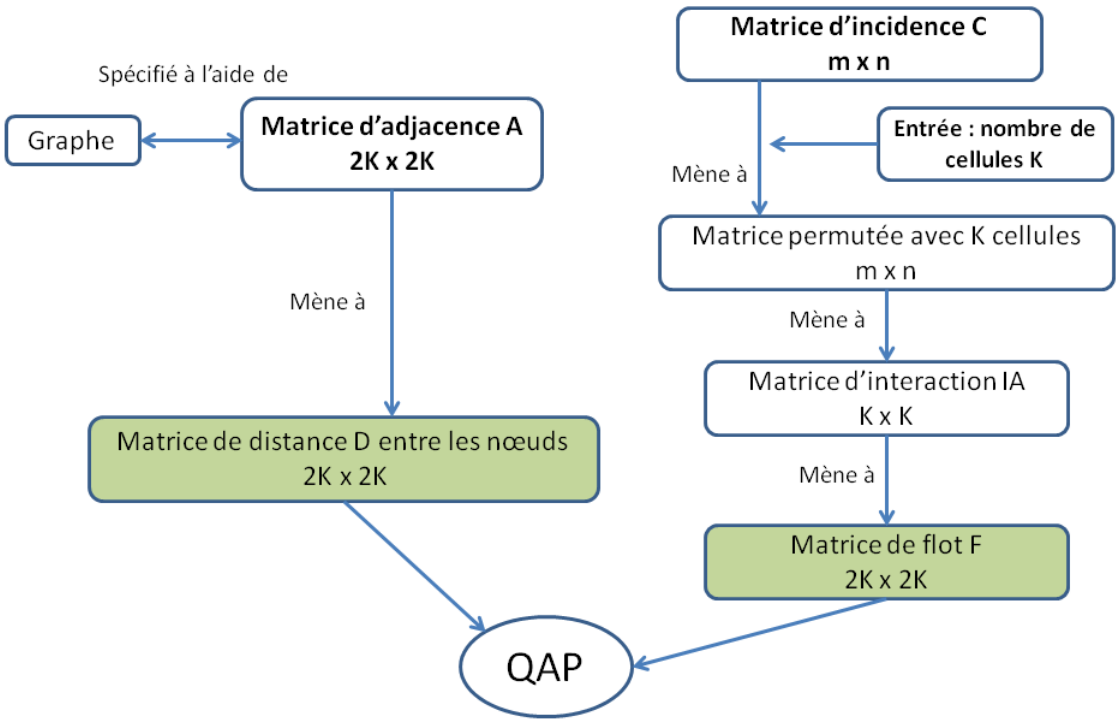


Figure 4.1 – Processus menant au problème d'affectation quadratique

Notons que le développement d'une méthode de résolution du QAP ne fait pas l'objet de ce mémoire. Nous nous intéressons plutôt à l'affectation des unités aux positions. C'est pour cette raison que nous avons utilisé l'algorithme de colonies de fourmis proposé par Taillard [56] pour résoudre le problème d'affectation quadratique.

4.1 Création de la matrice permutée avec K cellules en résolvant un problème de formation de cellules

Dans cette section et dans un souci de clarté, nous allons appuyer nos propos à l'aide d'exemples concrets. Nous partons donc d'une matrice d'incidence C de taille $m \times n$ telle que définie à la Figure 4.2. Les lignes i correspondent aux processeurs et les co-

		Mémoires										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Processeurs	**1**	1	1	0	0	0	1	0	0	0	0	0
	2	0	1	0	0	0	1	0	0	1	0	0
	3	1	0	1	0	0	0	1	0	0	0	1
	4	0	0	1	0	0	0	1	0	0	0	0
	5	0	0	1	1	0	0	0	0	0	0	1
	6	0	0	0	1	1	0	0	0	0	1	0
	7	0	0	0	0	1	0	0	1	0	1	0

Figure 4.2 – Matrice d'incidence C

lonnes j aux mémoires, de sorte que c_{ij} représente l'interaction entre le processeur i et la mémoire j. Cette matrice d'incidence est une donnée d'entrée, de même que le nombre de cellules K.

Dans l'exemple de la Figure 4.2, pour simplifier la suite des explications, lorsqu'un processeur accède à une mémoire, il n'y accède qu'une fois. Ceci explique pourquoi la matrice est constituée exclusivement de 0 et de 1.

Formulation en un problème de programmation en nombres entiers L'objectif est de générer une matrice permutée précisant les interactions entre les groupes de processeurs et les familles de mémoires. Le modèle doit maximiser la performance du réseau en réduisant les communications inter-cellulaires et en augmentant les communications intra-cellulaires tout en équilibrant la charge des cellules. Pour construire ce regroupement, nous formulons un problème de programmation mathématique en introduisant d'abord les variables binaires suivantes :

pour chaque paire $i = 1,\ldots,m;\ k = 1,\ldots,K$

$$x_{ik} = \begin{cases} 1 & \text{si le processeur } i \text{ appartient à la cellule } k \\ 0 & \text{sinon} \end{cases}$$

pour chaque paire $j = 1,\ldots,n;\ k = 1,\ldots,K$

$$y_{jk} = \begin{cases} 1 & \text{si la mémoire } j \text{ appartient à la cellule } k \\ 0 & \text{sinon} \end{cases}$$

Nous pouvons donc dire que :

$$x_{ik}y_{jk}c_{ij} = \begin{cases} c_{ij} & \text{si } i \text{ et } j \text{ appartiennent à la cellule } k \\ 0 & \text{sinon} \end{cases}$$

La fonction économique intègre deux objectifs que sont l'autonomie et le facteur de balance. Pour mesurer l'autonomie d'une solution, nous utilisons la formule suivante :

$$\text{Facteur d'autonomie} = \frac{c^{In}_{\neq 0}}{c} = \frac{\sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ik}y_{kj}}{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}} \tag{4.1}$$

où $c = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}$ dénote la somme de tous les éléments de la matrice A, et $c^{In}_{\neq 0}$ représente la somme des c_{ij} non nuls à l'intérieur des cellules.

Pour spécifier le facteur de balance, nous notons $T = \frac{c}{K}$ le nombre moyen d'opérations désirées dans chaque cellule. Le facteur de balance est alors exprimé ainsi :

$$\text{Facteur de balance} = \sum_{k=1}^{K}\left| T - \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ik}y_{jk}c_{ij} \right| \tag{4.2}$$

C'est en fait la somme des déviations dans les cellules, par rapport à la valeur moyenne.

La fonction objectif a pour but de maximiser l'efficacité définie comme suit :

$$\text{Eff } = (\text{Facteur d'autonomie}) - \frac{\gamma}{c}(\text{Facteur de balance}) \tag{4.3}$$

où γ est le terme qui spécifie l'importance que nous voulons donner à un facteur par

rapport à l'autre.

Le modèle complet est formulé comme un problème de programmation en nombres entiers où la fonction économique comporte des termes quadratiques :

$$\text{Max Eff} = \frac{\sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ik}y_{kj} - \gamma\sum_{k=1}^{K}\left|T - \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ik}y_{jk}c_{ij}\right|}{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}} \tag{4.4}$$

sujet à :

$$\sum_{k=1}^{K} x_{ik} = 1,\ \forall i \in \{1,\ldots,m\} \tag{4.5}$$

$$\sum_{k=1}^{K} y_{jk} = 1,\ \forall j \in \{1,\ldots,n\} \tag{4.6}$$

$$\sum_{i=1}^{m} x_{ik} \geq 1,\ \forall k \in \{1,\ldots,K\} \tag{4.7}$$

$$\sum_{j=1}^{n} y_{jk} \geq 1,\ \forall k \in \{1,\ldots,K\} \tag{4.8}$$

$$x_{ik} = 0\ ou\ 1\ \forall i\ \in \{1,\ldots,m\};\ \forall k\ \in \{1,\ldots,K\} \tag{4.9}$$

$$y_{jk} = 0\ ou\ 1\ \forall j\ \in \{1,\ldots,n\};\ \forall k\ \in \{1,\ldots,K\} \tag{4.10}$$

Les contraintes (4.5) et (4.6) assurent que chaque processeur et chaque mémoire sont assignés à exactement une cellule. Les contraintes (4.7) et (4.8) assurent que chaque cellule contient au moins un processeur et une mémoire. Enfin, les variables sont binaires dans (4.9) et (4.10).

Illustration sur un exemple À titre d'exemple, nous allons nous baser sur la matrice d'incidence de la Figure 4.2, qui est binaire pour des soucis de clarté. Nous pouvons y lire par exemple que le processeur 6 utilise les mémoires 4, 5 et 10.

En choisissant d'avoir 3 cellules, et en prenant $\gamma = 0.5$, nous obtenons la matrice permutée donnée à la Figure 4.3 en utilisant une méthode de résolution du problème de

formation de cellules proposée dans le mémoire de maîtrise de Ahadri [2].

		Mémoires										
		4	**5**	**8**	**10**	**1**	**2**	**6**	**9**	**3**	**7**	**11**
Processeurs	**6**	1	1	0	1	0	0	0	0	0	0	0
	7	0	1	1	1	0	0	0	0	0	0	0
	1	0	0	0	0	1	1	1	0	0	0	0
	2	0	0	0	0	0	1	1	1	0	0	0
	3	0	0	0	0	1	0	0	0	1	1	1
	4	0	0	0	0	0	0	0	0	1	0	0
	5	1	0	0	0	0	0	0	0	1	1	1

Figure 4.3 – **Exemple 1 :** Matrice permutée (3 cellules et $\gamma = 0.5$)

Le total des interactions dans la première cellule est la somme de ses éléments, c'est-à-dire 6. De même, les cellules 2 et 3 ont un nombre total d'interactions de 6 et 7 respectivement.

Les éléments non nuls à l'extérieur des cellules sont les éléments exceptionnels (indiqués en rouge). La somme des éléments exceptionnels est donc de 2 pour notre exemple. Nous pouvons aussi calculer la cible dans chaque cellule : $T = \dfrac{c}{K} = \dfrac{21}{3} = 7$. L'efficacité pour cette représentation est donc :

$$\begin{aligned} \text{Eff} &= \frac{c_1^{In}}{c} - \frac{\gamma}{c} \sum_{k=1}^{K} \left| T - \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ik} y_{jk} c_{ij} \right| \\ &= \frac{(6+6+7)}{21} - \frac{0.5}{21} \times ((7-6)+(7-6)+(7-7)) = 0.8571 \end{aligned} \tag{4.11}$$

Nous comprenons bien que suivant la valeur de γ, l'efficacité change. Si nous choisissons γ proche de 0, nous voulons peu d'éléments exceptionnels ; si en revanche nous choisissons γ proche de 1, nous privilégions le fait que les cellules soient équilibrées.

L'algorithme implémenté pour résoudre ce problème [20] dépasse le cadre de ce mémoire, toute l'information associée est cependant disponible dans le travail de Ahadri [2].

4.2 Création de la matrice d'interaction *IA*

Maintenant que nous avons la matrice permutée, nous pouvons définir une matrice d'interaction *IA* associée entre les groupes de processeurs et les familles de mémoires. En se référant à notre exemple, $K = 3$, nous distinguons alors les cellules 1, 2 et 3, ainsi que les groupes de processeurs et les familles de mémoires suivants, conformément à la Figure 4.3 :

```
Groupe_Processeur1 = {6,7}
Groupe_Processeur2 = {1,2}
Groupe_Processeur3 = {3,4,5}

Famille_Mémoire1 = {4,5,8,10}
Famille_Mémoire2 = {1,2,6,9}
Famille_Mémoire3 = {3,7,11}
```

La matrice *IA* correspondante est présentée Figure 4.4. L'élément à l'intersection de la ligne P_r et de la colonne M_s représente l'interaction entre les processeurs du groupe P_r et les mémoires de la famille M_s.

	M1	**M**2	**M**3
P1	6	0	0
P2	0	6	0
P3	1	1	7

Figure 4.4 – **Exemple 1 :** Matrice d'interaction *IA*

4.3 Affectation sans prise en compte de l'importance thermique

Dans cette partie, nous allons résoudre le problème de l'affectation des composantes, c'est-à-dire des groupes de processeurs et des familles de mémoires, aux différents nœuds du graphe $G = (V, E)$ de l'architecture constitué d'un ensemble de nœuds V et d'un ensemble d'arêtes E. Rappelons que le problème (QAP) requiert en entrée une

matrice de flot F et une matrice de distance D afin d'être formulé comme suit :

$$\bar{z} = \min_{\phi} \sum_{i=1}^{2K} \sum_{j=1}^{2K} d_{ij} f_{\phi(i)\phi(j)} \qquad \phi : \text{permutation de } \{1, \cdots, 2K\}$$

Détermination de la matrice de flot Nous utilisons la matrice d'interaction IA pour construire la matrice de flot F dont les entrées représentent les interactions entre les composantes (groupes de processeurs et familles de mémoires). Les processeurs ne sont pas tenus d'interagir entre eux, et il en est de même pour les mémoires. C'est ainsi que nous pouvons construire la matrice de flot F de la façon suivante :

$$F = \left[\begin{array}{cccc|cccc} 0 & 0 & \dots & 0 & IA_{P_1M_1} & IA_{P_1M_2} & \dots & IA_{P_1M_K} \\ 0 & 0 & \dots & 0 & IA_{P_2M_1} & IA_{P_2M_2} & \dots & IA_{P_2M_K} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & IA_{P_KM_1} & IA_{P_KM_2} & \dots & IA_{P_KM_K} \\ \hline IA_{P_1M_1} & IA_{P_1M_2} & \dots & IA_{P_1M_K} & 0 & 0 & \dots & 0 \\ IA_{P_2M_1} & IA_{P_2M_2} & \dots & IA_{P_2M_K} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ IA_{P_KM_1} & IA_{P_KM_2} & \dots & IA_{P_KM_K} & 0 & 0 & \dots & 0 \end{array}\right]$$

En se référant à notre exemple, une telle matrice est illustrée à la Figure 4.5 pour représenter les interactions entre les groupes de processeurs P_1, P_2, P_3, et les familles de mémoires M_1, M_2 et M_3.

	P1	**P**2	**P**3	**M**1	**M**2	**M**3
P1	0	0	0	6	0	0
P2	0	0	0	0	6	0
P3	0	0	0	1	1	7
M1	6	0	1	0	0	0
M2	0	6	1	0	0	0
M3	0	0	7	0	0	0

Figure 4.5 – **Exemple 1 :** Matrice de flot F

Différentes représentations de la matrice de flot F peuvent être utilisées, comme expliqué par Élie [20] : nous pouvons notamment empêcher deux processeurs ou deux mémoires d'être assignés à des nœuds adjacents en introduisant des flots fortement négatifs dans la matrice de flot. Cependant, dans le cadre de ce mémoire, nous avons choisi que le flot entre deux processeurs ou deux mémoires est nul.

Détermination de la matrice de distance Nous allons partir de la matrice d'adjacence du graphe de l'architecture pour déterminer la matrice de distance. En effet, pour obtenir la matrice de distance D, nous utilisons l'algorithme de Floyd-Warshall [25] qui permet de déterminer les plus courts chemins dans un graphe.

Algorithme 1 : Algorithme de Floyd-Warshall.

Entrées : A (matrice d'adjacence $n \times n$)
Sorties : D (matrice $n \times n$)

$D \leftarrow A$
pour $i \leftarrow 1$ **à** n **faire**
  pour $j \leftarrow 1$ **à** n **faire**
    si $D(i,j) = 0$ **alors**
      $D(i,j) \leftarrow +\infty$
    fin
  fin
fin
pour $k \leftarrow 1$ **à** n **faire**
  pour $i \leftarrow 1$ **à** n **faire**
    pour $j \leftarrow 1$ **à** n **faire**
      $D(i,j) \leftarrow \min[D(i,j), D(i,k) + D(k,j)]$
    fin
  fin
fin
pour $i \leftarrow 1$ **à** n **faire**
  $D(i,i) \leftarrow 0$
fin
retourner D

Notons que la matrice d'adjacence prise en entrée doit être légèrement modifiée, comme indiqué dans la première partie de l'algorithme. En effet, pour que l'algorithme

fonctionne, il faut qu'elle soit construite comme suit :

$$A(i,j) = \begin{cases} 1 & \text{si les nœuds } i \text{ et } j \text{ sont reliés} \\ +\infty & \text{sinon} \end{cases}$$

L'algorithme de Floyd-Warshall génère donc une matrice D où chaque élément d_{ij} représente la longueur du chemin le plus court entre le nœud i et le nœud j. De plus, l'algorithme permet d'identifier explicitement le chemin le plus court qui sera utilisé pour acheminer le flot entre les nœuds i et j. Notons que la matrice de distance D sera une entrée du problème d'affectation quadratique.

4.3.1 Illustration avec un exemple : définition des matrices

Dans cette section, nous nous appuyons sur un problème concret afin d'illustrer plus clairement le processus.

- nous prendrons la matrice de flot F avec 8 nœuds de la Figure 4.7, qui a été obtenue à partir de la matrice d'interaction IA donnée à la Figure 4.6.

	M1	M2	M3	M4
P1	87	0	1	0
P2	1	48	5	7
P3	9	6	27	1
P4	7	0	5	54

Figure 4.6 – **Exemple 2 :** Matrice d'interaction IA

	P1	P2	P3	P4	M1	M2	M3	M4
P1	0	0	0	0	87	0	1	0
P2	0	0	0	0	1	48	5	7
P3	0	0	0	0	9	6	27	1
P4	0	0	0	0	7	0	5	54
M1	87	1	9	7	0	0	0	0
M2	0	48	6	0	0	0	0	0
M3	1	5	27	5	0	0	0	0
M4	0	7	1	54	0	0	0	0

Figure 4.7 – **Exemple 2 :** Matrice de flot F associée à la matrice d'interaction

- pour la matrice de distance, nous allons partir de la représentation du graphe de la Figure 4.8 dont la matrice d'adjacence A est illustrée à la Figure 4.9.

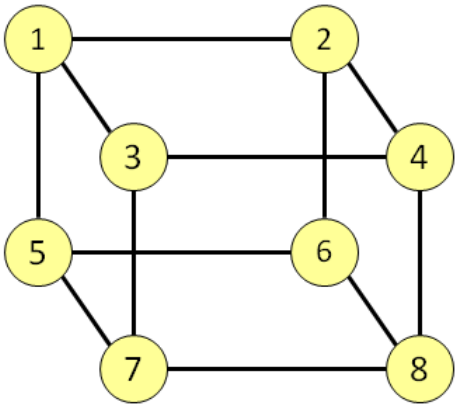


Figure 4.8 – **Exemple 2 :** Graphe à 8 nœuds

	1	2	3	4	5	6	7	8
1	0	1	1	0	1	0	0	0
2	1	0	0	1	0	1	0	0
3	1	0	0	1	0	0	1	0
4	0	1	1	0	0	0	0	1
5	1	0	0	0	0	1	1	0
6	0	1	0	0	1	0	0	1
7	0	0	1	0	1	0	0	1
8	0	0	0	1	0	1	1	0

Figure 4.9 – **Exemple 2 :** Matrice d'adjacence A

Notons que les réseaux d'architecture multiprocesseurs sont particuliers dans le sens où nous ne retrouvons pas d'arêtes « diagonales » du type $(1,6)$ ou $(2,7)$ dans le graphe de la Figure 4.8. En fait, les réseaux d'architecture multiprocesseurs sont obtenus en accolant des blocs dont la représentation est illustrée à la Figure 4.8. Par exemple, nous verrons à la section 4.4.2 que le réseau de la Figure 4.13 est obtenu en accolant 4 blocs simples. Remarquons que comme le graphe est non orienté, la matrice d'adjacence est symétrique.

Pour obtenir la matrice de distance D, nous utilisons l'algorithme de Floyd-Warshall appliqué sur la matrice d'adjacence de la Figure 4.9. Nous disposons alors d'une matrice de distance D, illustrée à la Figure 4.10, qui constitue une entrée du problème d'affectation quadratique.

4.3.2 Illustration avec cet exemple : la solution obtenue du problème d'affectation quadratique

La résolution du problème d'affectation quadratique à l'aide de l'algorithme de colonies de fourmis va nous permettre d'affecter les quatre groupes de processeurs et les quatre familles de mémoires aux différents nœuds du graphe, de façon à minimiser la

	1	2	3	4	5	6	7	8
1	0	1	1	2	1	2	2	3
2	1	0	2	1	2	1	3	2
3	1	2	0	1	2	3	1	2
4	2	1	1	0	3	2	2	1
5	1	2	2	3	0	1	1	2
6	2	1	3	2	1	0	2	1
7	2	3	1	2	1	2	0	1
8	3	2	2	1	2	1	1	0

Figure 4.10 – **Exemple 2 :** Matrice de distance D

somme des produits *distance* x *flot*. Il s'agit de calculer la quantité suivante :

$$\bar{z} = \min_{\phi} \sum_{i=1}^{8} \sum_{j=1}^{8} d_{ij} f_{\phi(i)\phi(j)} \qquad \phi : \text{permutation de } \{1, \cdots, 8\}$$

où les matrices F et D sont définies respectivement aux Figures 4.7 et 4.10. Après avoir appliqué 200 itérations[1] de l'algorithme de colonies de fourmis, nous obtenons la permutation ϕ suivante :

$$\phi = [5\ 4\ 3\ 7\ 1\ 8\ 6\ 2]$$

Notons que $\phi(i)$ correspond à la composante affectée au nœud i, en sachant que les composantes sont numérotées en suivant l'ordre $P_1, P_2, \cdots, P_K, M_1, M_2, \cdots, M_K$. Le Tableau 4.11 résume l'affectation que nous pouvons tirer de la permutation ϕ.

Numéro de nœud	1	2	3	4	5	6	7	8
Numéro de composante	5	4	3	7	1	8	6	2
Composante	M1	P4	P3	M3	P1	M4	M2	P2

Figure 4.11 – Interprétation de la permutation ϕ

La valeur de la fonction objectif $\bar{z}$ associée est :

$$\bar{z} = \sum_{i=1}^{8} \sum_{j=1}^{8} d_{ij} f_{\phi(i)\phi(j)} = 528$$

[1] Pour un problème de cette taille, 200 itérations sont suffisantes pour aboutir à la solution optimale.

Le résultat graphique de ce problème d'affectation quadratique est présenté à la Figure 4.12. Pour ce qui est de la répartition des flots sur les arêtes, rappelons que c'est l'algorithme de Floyd-Warshall qui définit les chemins les plus courts entre chaque paire de nœuds. Nous connaissons le flot demandé entre chaque paire de composantes grâce à la matrice de flot F, et la permutation ϕ indique à quel nœud chaque composante est affectée. Nous pouvons alors déterminer les flots dans la structure. D'autre part, précisons que le flot indiqué sur une arête correspond au flot qui passe dans cette arête mais ce flot peut desservir plusieurs nœuds.

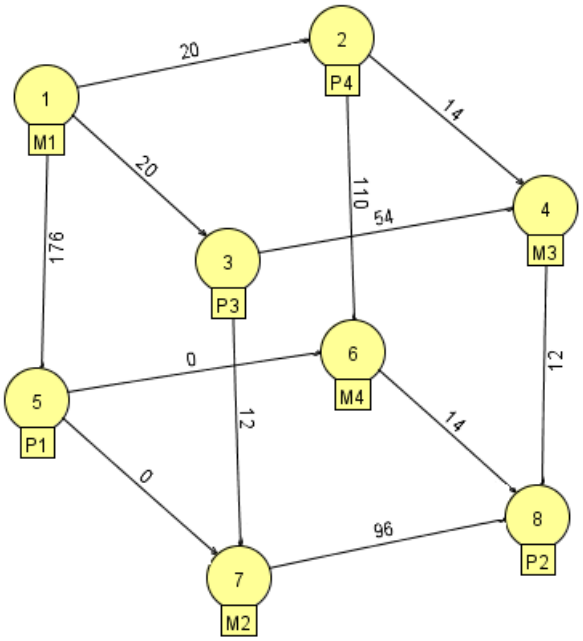


Figure 4.12 – **Exemple 2 :** Résultat du problème d'affectation quadratique

Dans la section suivante, en utilisant un exemple de plus grande taille, nous allons illustrer le fait que pour prendre en compte le dégagement de chaleur et éviter des flots importants au centre de la structure, nous allons devoir modifier le modèle.

4.4 Deuxième affectation : prise en compte de l'importance thermique

4.4.1 Explication du phénomène thermique

La circulation d'un flot d'interaction sur une arête du réseau entraîne un dégagement de chaleur. Les systèmes électroniques actuels sont très souvent équipés de ventilateurs aux extrémités de la structure. Parfois même nous utilisons des dissipateurs de chaleur. Ces dispositifs permettent d'évacuer les pertes dissipées par les éléments qui produisent de la chaleur. Le matériau utilisé a son importance (comme l'aluminium) et la surface de contact entre le dissipateur et le composant doit être la plus grande possible pour maximiser son efficacité.

Cependant, le cœur d'un composant est souvent difficile à atteindre et pour ce qui nous concerne, le problème se situe au niveau des arêtes centrales où la chaleur va se dissiper à l'intérieur de la structure, réchauffant ainsi les autres composantes et provoquant une surchauffe dangereuse pour le système. Intuitivement, si nous ne prenons pas en compte le dégagement de chaleur, nous nous doutons que le problème d'affectation quadratique va générer une solution où le centre de la structure sera un pôle très dense au niveau des communications, puisque le cœur de la structure est, par définition, accessible facilement par les composantes. Nous y retrouverons ainsi des composantes qui agissent avec beaucoup d'autres. En conséquence, les arêtes centrales risquent d'être très utilisées puisqu'elles se retrouvent sur le plus court chemin entre plusieurs composantes.

L'idée est alors de repérer ces arêtes centrales et de pénaliser le passage du flot sur ces arêtes, pour rediriger ce dernier en périphérie du système. Pour ce faire, nous allons modifier la matrice de distance.

4.4.2 Modification de la matrice de distance

Dans cette section, nous allons illustrer notre propos sur un exemple de taille plus grande que précédemment, pour mieux comprendre le dégagement de chaleur.

- nous prendrons la matrice de flot F avec 18 nœuds (9 cellules) disponible à l'Annexe I

- pour la matrice de distance, nous allons partir de la représentation du graphe de la Figure 4.13 dont la matrice d'adjacence A correspondante est disponible à l'Annexe II

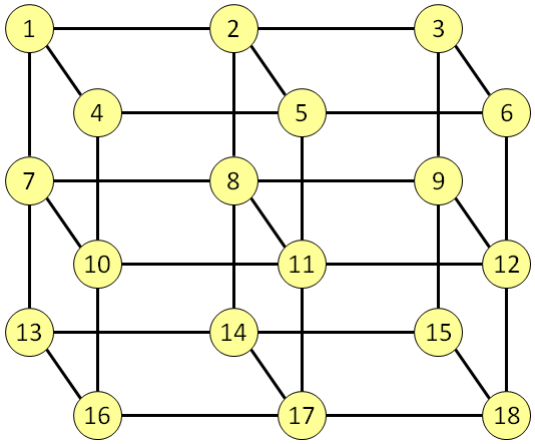


Figure 4.13 – **Exemple 3 :** Graphe à 18 nœuds

L'idée est de partir de la matrice d'adjacence A pour aboutir à une matrice de distance $D_{modifiee}$ qui sera donnée en entrée du problème d'affectation quadratique. Pour calculer la matrice $D_{modifiee}$, nous allons suivre le processus indiqué à la Figure 4.14.

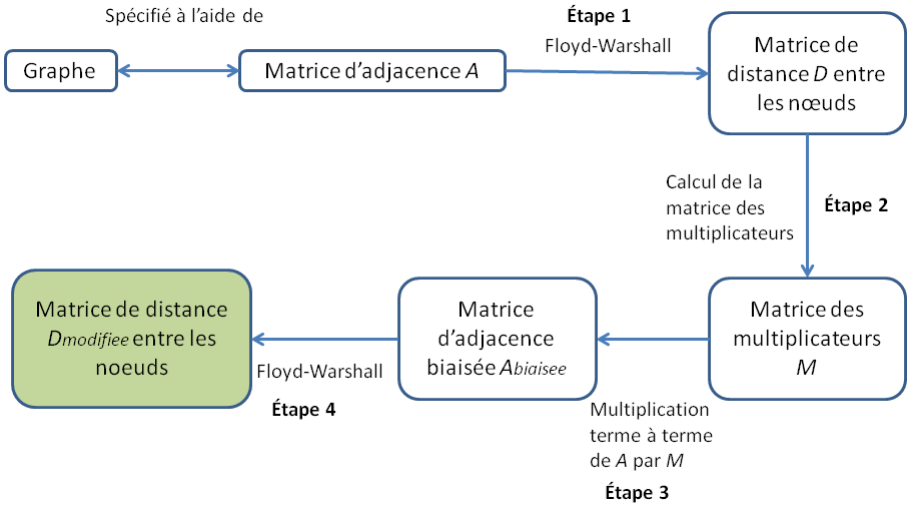


Figure 4.14 – **Exemple 3 :** Processus menant à la matrice $D_{modifiee}$

Étape 1 Avant de déterminer la matrice de distance $D_{modifiee}$ à fournir au problème d'affectation quadratique, déterminons les arêtes centrales. Dans un premier temps, appliquons l'algorithme de Floyd-Warshall sur la matrice d'adjacence A. Nous allons donc connaitre la longueur des plus courts chemins entre les nœuds du graphe. Notons D cette matrice de distance après application de l'algorithme de Floyd-Warshall (Annexe III).

Étape 2 Il s'agit ici de déterminer la matrice des multiplicateurs M qui permet de pondérer chaque arête en fonction de sa situation dans le graphe. Notons $d_{i\bullet} = \sum_{j=1}^{n} d_{ij}$ la somme des distances du nœud i aux autres nœuds du graphe. Nous pouvons vérifier que plus cette valeur est petite, plus le nœud i est central. Nous pouvons par exemple voir sur le graphe de la Figure 4.13 deux nœuds centraux : 8 et 11.

Nous supposons que la chaleur produite au centre de la structure va se dissiper dans tous les sens dans la structure pour finir par s'échapper en dehors de celle-ci. L'idée est alors de rediriger les flots pour éviter une trop grande concentration au centre de la structure en pénalisant l'utilisation des chemins y passant. Notons que nous n'avons pas pu obtenir des informations pertinentes sur le dégagement de chaleur et sur une formulation mathématique pour le caractériser. C'est pour cette raison que nous avons utilisé la relation suivante pour modifier la matrice de distance à l'aide de la matrice des multiplicateurs M permettant d'indiquer les arêtes les plus centrales :

$$M_{ij} = \exp\left(\frac{1}{d_{i\bullet} + d_{j\bullet}}\right)$$

Plus $d_{i\bullet}$ et $d_{j\bullet}$ sont grands, plus M_{ij} sera petit. L'exponentielle symbolise le dégagement progressif de la chaleur. Notons que la matrice M sera normalisée de telle sorte que tous ses termes seront compris entre 1 et 100. Nous illustrons à la Figure 4.15 les arêtes les plus centrales dans la structure.

Plus le multiplicateur associé à une arête est important, plus l'arête est épaisse. Par exemple, l'arête $(8, 11)$ est associée au multiplicateur 100. Notons que les arêtes de même couleur ont exactement le même multiplicateur. Dans notre exemple, il y a cinq couleurs différentes.

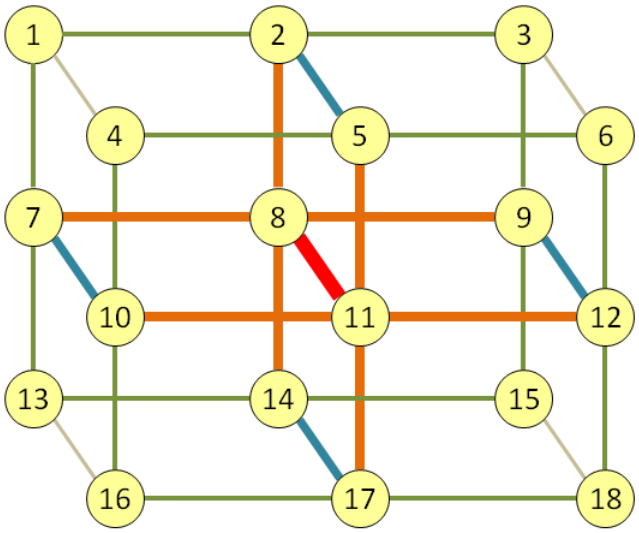


Figure 4.15 – **Exemple 3 :** Illustration de la matrice des multiplicateurs

Étape 3 Pour obtenir une nouvelle matrice d'adjacence biaisée $A_{biaisee}$, nous multiplions terme à terme la matrice d'adjacence A par la matrice des multiplicateurs M.

Étape 4 Nous devons maintenant réappliquer l'algorithme de Floyd-Warshall sur la matrice $A_{biaisee}$, car les chemins les plus courts ont sûrement changé, étant donné que les distances entre les nœuds ne sont plus les mêmes. Nous obtenons alors la matrice de distance $D_{modifiee}$ qui sera utilisée en entrée du problème d'affectation quadratique (voir Annexe IV).

Sur le graphe de la Figure 4.13, nous pouvons voir que l'arête la plus centrale est celle reliant les nœuds 8 et 11. Il est donc logique que la distance entre le nœud 8 et le nœud 11 soit très grande dans la nouvelle matrice de distance $D_{modifiee}$. C'est ce que nous pouvons vérifier à l'Annexe IV.

4.4.3 Solution du problème d'affectation quadratique avec $D_{modifiee}$

Maintenant que nous avons modifié la matrice de distance, nous pouvons la fournir en entrée de l'algorithme en charge de résoudre le problème d'affectation quadratique.

Avec la matrice $D_{modifiee}$ et la matrice de flot F, l'algorithme va nous permettre

d'affecter les neuf groupes de processeurs et les neuf familles de mémoires aux différents nœuds du graphe, de façon à minimiser la somme des produits *distance* x *flot*. Après avoir appliqué 500 itérations[2] de l'algorithme de colonies de fourmis, nous obtenons les résultats suivants :

$$\phi = [18\ 7\ 14\ 9\ 1\ 5\ 16\ 6\ 2\ 10\ 3\ 12\ 17\ 15\ 11\ 8\ 13\ 4]$$

$$\bar{z} = \sum_{i=1}^{18} \sum_{j=1}^{18} d_{biaisee_{ij}} f_{\phi(i)\phi(j)} = 32094$$

Encore une fois, notons que le flot entre chaque paire de composantes utilise le chemin le plus court identifié avec l'algorithme de Floyd-Warshall. Le résultat graphique de l'affectation et de la répartition des flots est présenté à la Figure 4.16. Nous pouvons y voir que le flot dans les arêtes centrales est faible, voire nul. Le chemin pour aller d'un nœud à un autre peut donc être plus long, mais en contrepartie, il ne passe pas au centre de la structure. De façon générale, les flots respectent en moyenne les contraintes imposées par la matrice $D_{modifiee}$, qui sont en accord avec la matrice des multiplicateurs M.

À titre de comparaison, nous avons résolu le problème d'affectation quadratique avec la matrice de distance originale, c'est-à-dire celle obtenue en appliquant l'algorithme de Floyd-Warshall sur la matrice d'adjacence A. Le résultat graphique obtenu en effectuant 500 itérations de l'algorithme de colonies de fourmis est présenté à la Figure 4.17. Contrairement au problème d'affectation quadratique biaisé, nous pouvons observer des flots importants au centre de la structure, puisqu'aucune contrainte ne redirige les flots vers l'extérieur.

[2]Pour un problème de cette taille, 500 itérations sont suffisantes pour aboutir à une solution de bonne qualité.

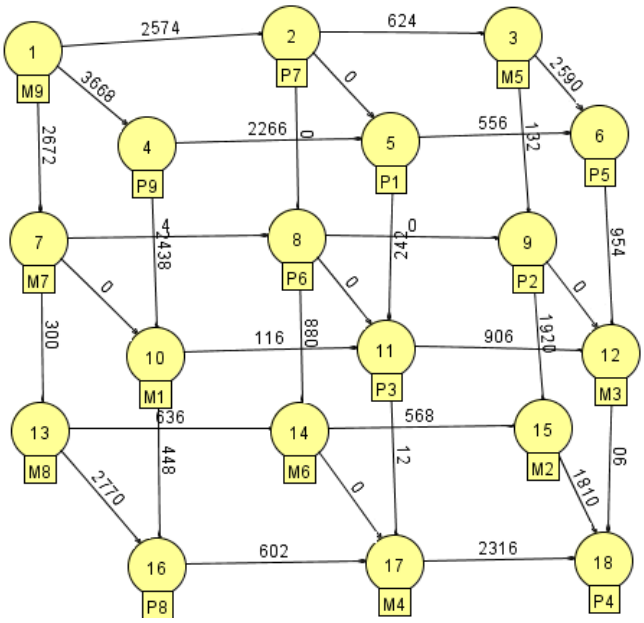


Figure 4.16 – **Exemple 3 :** Résultat du problème d'affectation quadratique biaisé avec 18 nœuds

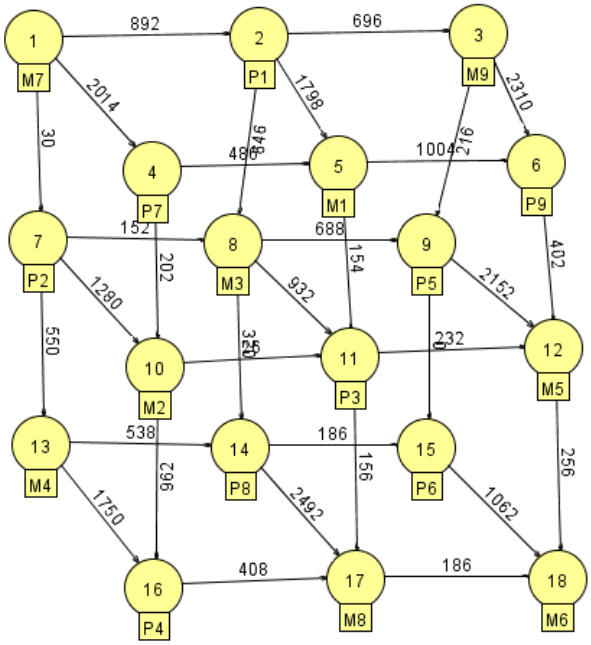


Figure 4.17 – **Exemple 3 :** Résultat du problème d'affectation quadratique avec 18 nœuds

Notons que sur ces graphes, le flot indiqué sur une arête correspond au flot qui passe dans cette arête mais ce flot peut desservir plusieurs nœuds. Par exemple, considérons l'arête $(12, 18)$ du graphe de la Figure 4.16. Le Tableau 4.I regroupe les chemins qui passent par cette arête et qui relient deux composantes qui interagissent entre eux, ainsi que le flot entre ces composantes.

Chemin	Départ	Arrivée	Flot demandé
1 2 3 6 12 18	M9	P4	15
18 12 6 3 2 1	P4	M9	15
6 12 18 17	P5	M4	9
17 18 12 6	M4	P5	9
11 12 18 15	P3	M2	13
15 18 12 11	M2	P3	13
12 18 17 16	M3	P8	1
16 17 18 12	P8	M3	1
12 18	M3	P4	7
18 12	P4	M3	7
			Total : 90

Tableau 4.I – **Exemple 3 :** Chemins empruntant l'arête $(12, 18)$ dans le graphe de la Figure 4.16

Remarquons que le caractère symétrique de la matrice de flot induit des chemins qui vont par paires, comme nous pouvons le constater dans le Tableau 4.I.

Nous venons donc de trouver un moyen d'affecter les composantes aux nœuds de la structure en résolvant un problème d'affectation quadratique où la matrice de distance entre les nœuds est modifiée en augmentant fictivement la longueur des arêtes centrales pour réduire le flot qui y circule et réduire l'effet du dégagement de chaleur. Cependant, les chemins entre chaque paire de nœuds sont toujours définis par l'algorithme de Floyd-Warshall qui ne fait qu'indiquer le chemin le plus court entre ces deux nœuds. Mais il n'y a pas unicité de ce chemin, et c'est ce que nous allons exploiter dans le problème de sélection des bus.

CHAPITRE 5

PROBLÈME DE SÉLECTION DES BUS

Le problème de sélection des bus est un problème d'affectation que nous résolvons après l'affectation des composantes aux nœuds de l'architecture. Pour chaque arête, nous disposons d'un ensemble de bus de différentes capacités et de différents coûts. Naturellement, le flot sur une arête ne peut pas dépasser la capacité du bus affecté à celle-ci. Le problème de sélection des bus vise à optimiser les flots en minimisant deux éléments : le coût total des bus affectés aux arêtes et le dégagement de chaleur associée à la structure en fonctionnement.

Dans un premier temps, nous allons présenter le modèle basé sur une hypothèse de configuration radiale, puis nous verrons que ce problème a des similarités avec celui de l'alimentation des clients à partir d'une station dans un réseau électrique. Nous utiliserons enfin ces similitudes pour définir notre méthode de résolution, et nous exposerons le pseudocode associé.

5.1 Énoncé du modèle

Rappelons que la résolution du problème d'affectation quadratique a permis d'assigner les composantes (groupes de processeurs et familles de mémoires) aux nœuds de la structure. Ensuite, le flot sur les arêtes est obtenu en dirigeant les flots sur les composantes le long des plus courts chemins identifiés avec l'algorithme de Floyd-Warshall. Dans le modèle qui suit, nous désirons rediriger les flots entre les composantes pour favoriser la sélection des bus, c'est-à-dire réduire le coût total des bus et réduire le dégagement de chaleur.

Nous allons définir notre modèle en faisant l'hypothèse suivante :

Hypothèse *Les interactions entre tout groupe de processeurs* P_k $(k = 1, \cdots, K)$ *et toute famille de mémoire* M_k $(k = 1, \cdots, K)$ *se font au travers d'un chemin unique*

dans la structure.

Cette hypothèse signifie donc que les interactions entre un groupe de processeurs $P_{k'}$ donné et les familles de mémoire M_k $(k = 1, \cdots, K)$ se font sur une configuration radiale qui correspond à un arbre dont la racine est le nœud auquel est affecté $P_{k'}$, comme l'illustrent les Figures 5.1 et 5.2. Dans le modèle mathématique qui suit, cette configuration radiale associée à $P_{k'}$ est notée $\Gamma^{k'}$.

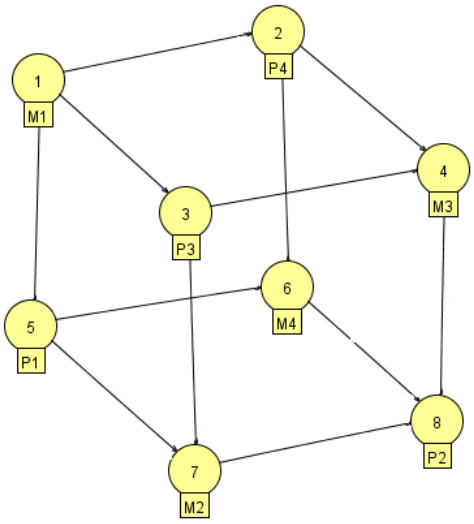


Figure 5.1 – Exemple de graphe à 8 nœuds

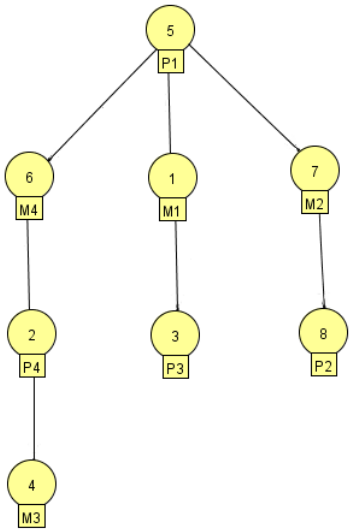


Figure 5.2 – Exemple d'arbre associé à P_1

Pour formuler notre problème, nous utiliserons les notations suivantes :

f_e^k : flot qui provient du processeur P_k et qui circule sur l'arête $e \in E$

E_v : ensemble des arêtes incidentes au nœud $v \in V$

Γ^k : configuration radiale des interactions associées au groupe de processeurs P_k

e_v^k : arête incidente à v sur le chemin entre P_k et v dans la configuration Γ^k

z_{ew} : variable binaire pour vérifier si le bus w est utilisé sur l'arête $e \in E$

W_e : ensemble des bus qui peuvent être utilisés sur l'arête $e \in E$

u_w : capacité du bus w

ς_{ew} : coût d'utilisation du bus w sur l'arête $e \in E$

Eng_e : dégagement de chaleur pour l'arête e

Pour simplifier la notation, nous dénotons le nœud où est affecté un groupe de processeurs P_k par le même symbole P_k.

Lors de la définition de la matrice des multiplicateurs M dans la section 4.4.2, nous avons vu que les arêtes centrales doivent être pénalisées car elles produisent de la chaleur qui va se dissiper dans le reste de la structure. Ainsi, nous définissons le dégagement de chaleur d'une arête comme suit :

$$Eng_e = M_e f_e$$

où $f_e = \sum_{k=1}^{K} f_e^k$ est le flot total qui traverse l'arête e et M_e est l'élément M_{ab} de la matrice M où e est l'arête reliant les sommets a et b.

L'absence d'information plus pertinente sur le dégagement de chaleur nous a conduit à utiliser la formulation précédente. Ce terme de la fonction économique pourra être modifié pour le rendre plus proche de la réalité sans pour autant modifier la méthode de résolution.

Le problème de la sélection des bus (BSP) est alors le suivant :

$$\min\ \theta = \alpha \sum_{e \in E} \sum_{w \in W_e} z_{ew} \varsigma_{ew} + (1-\alpha) \sum_{e \in E} Eng_e$$

Sujet à

Pour chaque groupe de processeurs $k = 1, 2, \cdots, K$:

$$\sum_{e \in E_{P_k}} f_e^k = \sum_{\bar{k}=1}^{K} IA_{k\bar{k}} \tag{5.1}$$

Pour chaque nœud v différent du nœud où P_k est assigné :

$$f_{e_v^k}^k - \sum_{\substack{e \in E_v \\ e \neq e_v^k}} f_e^k = \begin{cases} IA_{k\bar{k}} & \text{si } M_{\bar{k}} \text{ est assigné à } v \\ \\ 0 & \text{sinon} \end{cases} \tag{5.2}$$

Les flots f_e^k suivent une configuration radiale Γ^k (5.3)

$$2 \sum_{k=1}^{K} f_e^k \leq \sum_{w \in W_e} z_{ew} u_w \quad e \in E \tag{5.4}$$

$$\sum_{w \in W_e} z_{ew} = 1 \qquad e \in E \tag{5.5}$$

$$z_{ew} = 0 \text{ ou } 1 \qquad w \in W_e,\ e \in E \tag{5.6}$$

$$f_e^k \geq 0, \text{ entier} \qquad e \in E,\ k = 1, 2, \cdots, K \tag{5.7}$$

où IA est la matrice d'interaction de taille $K \times K$ spécifiant le flot entre les groupes de processeurs et les familles de mémoires.

L'équation (5.1) assure l'interaction totale de P_k vers l'ensemble des familles de mémoires. L'équation (5.2) garantit la conservation du flot. L'équation (5.4) indique que le flot total circulant sur une arête donnée ne peut dépasser la capacité du bus utilisé sur cette arête. L'équation (5.5) stipule que pour une arête donnée, un seul bus est utilisé. Finalement le paramètre $\alpha \in [0,1]$ dans la fonction économique indique le poids relatif désiré entre le coût des bus et celui du dégagement de chaleur.

5.2 Analogie avec le problème des arcs d'alimentation

Ce problème est assez similaire au problème de la sélection des arcs d'alimentation dans un réseau électrique [45], [46] où les clients sont alimentés à partir d'une station. Les clients peuvent être reliés entre eux ou directement à une station. De plus, dans ce problème nous faisons l'hypothèse que chaque client est alimenté par un chemin unique à partir de la station en passant possiblement par la localisation d'autres clients. Le réseau de distribution a une topologie radiale donnant lieu à des arbres de recouvrement orientés, dont chaque racine correspond à une station. Dans ce cas, l'objectif est de déterminer les arcs de distribution et la capacité des conducteurs dans le but de minimiser le coût total des conducteurs et les pertes d'énergie le long du réseau de distribution.

Les résultats obtenus par Parada *et al.* [46] ont montré que l'heuristique la plus efficace pour ce genre de problème est la recherche avec tabous. C'est la raison pour laquelle nous avons décidé d'utiliser cette méthode pour résoudre notre problème.

5.3 Définition des points clés de la recherche avec tabous

Pour développer notre méthode de recherche avec tabous, nous avons besoin de définir quatre grandes notions qui permettent le bon déroulement de ce processus :

- une solution initiale qui servira de base à la recherche
- un voisinage qui défini l'ensemble des mouvements applicables
- une liste de tabous dont nous préciserons la taille
- la notion d'aspiration utilisée dans le processus de recherche

Nous allons dans cette section détailler les choix effectués quant à la définition de ces quatre éléments.

5.3.1 Solution initiale

Nous distinguons deux approches pour définir une solution initiale : la première se base sur les résultats obtenus lors de la résolution du problème (QAP) alors que la

deuxième relève d'une génération aléatoire. Comme nous avons formulé une hypothèse de radialité, une solution réalisable (en particulier la solution initiale) peut être définie par K arbres associés aux K groupes de processeurs.

Utilisation de l'algorithme de Floyd-Warshall La solution obtenue en dirigeant les flots entre les composantes le long des chemins identifiés avec l'algorithme de Floyd-Warshall lors de la résolution du problème d'affectation quadratique peut être utilisée comme solution initiale. En effet, l'algorithme de Floyd-Warshall définit un chemin unique entre toutes les paires de nœuds du graphe. Nous pouvons ainsi utiliser ces chemins pour définir les arbres initiaux.

Création d'une solution aléatoire Nous pouvons aussi choisir de partir d'une solution initiale quelconque, en utilisant par exemple un algorithme de génération d'arbres couvrants aléatoires. Pour obtenir un arbre pour chaque groupe P_k, $k \in 1, \cdots, K$, nous utiliserons l'algorithme de Wilson [64], qui est très simple à implémenter et en même temps très rapide.

Considérons un graphe à n nœuds. L'algorithme prend en entrée une racine (entier compris entre 1 et n) et renvoie un arbre défini par un vecteur noté $Next$ comportant n éléments. Pour chaque nœud i $(i = 1, \cdots, n)$, $Next(i)$ représente le père du nœud i. Le père de la racine est noté 0, nous avons donc $Next(racine) = 0$.

La méthode $VoisinAleatoire(u)$ qui apparaît dans l'Algorithme 2 renvoie un voisin choisi au hasard parmi les voisins du nœud u. Pour illustrer cet algorithme, considérons le graphe de la Figure 5.3 comportant 8 nœuds. Un exemple d'arbre aléatoire dont la racine est le nœud associé à P_1 est illustré à la Figure 5.4. Cet arbre correspond au vecteur $Next$ suivant :

$$Next = [5\ 6\ 1\ 2\ 0\ 5\ 5\ 7]$$

De la même manière, nous créons alors K arbres associés à $P_1, P_2, \cdots, P_K$. Ensuite, il suffit de déterminer le flot sur chacune des arêtes de la structure en utilisant les équations (5.1) et (5.2).

Algorithme 2 : Algorithme de Wilson.

Entrées : *racine* (entier définissant la racine de l'arbre)
Sorties : *Next* (vecteur de taille *n* représentant l'arbre créé)

pour $i \leftarrow 1$ **à** n **faire**
  $inTree(i) \leftarrow$ false
fin

$Next(racine) \leftarrow 0$
$InTree(racine) \leftarrow true$

pour $i \leftarrow 1$ **à** n **faire**
  $u \leftarrow i$
  tant que $InTree(u) = false$ **faire**
    $Next(u) \leftarrow VoisinAleatoire(u)$
    $u \leftarrow Next(u)$
  fin
  $u \leftarrow i$
  tant que $InTree(u) = false$ **faire**
    $InTree(u) \leftarrow true$
    $u \leftarrow Next(u)$
  fin
fin

retourner *Next*

Dans la section 6, nous comparerons numériquement les résultats obtenus avec ces deux types de solutions initiales : celle obtenue lors de la résolution du problème d'affectation quadratique utilisant l'algorithme de Floyd-Warshall, et celle obtenue par génération d'arbres couvrants aléatoires avec l'algorithme de Wilson.

5.3.2 Voisinage

Avant de définir le voisinage d'une solution courante, introduisons la notion de mouvement illustrée à la Figure 5.5.

Mouvement (Définition) *Considérons l'arbre des chemins associé à un groupe de processeurs P_k. Un mouvement (P_k, n_1, pn_1, n_2) est spécifié à l'aide d'un triplet (n_1, pn_1, n_2) où le nœud n_1 est décroché de son père pn_1 dans l'arbre pour le raccrocher plutôt à un autre sommet n_2.*

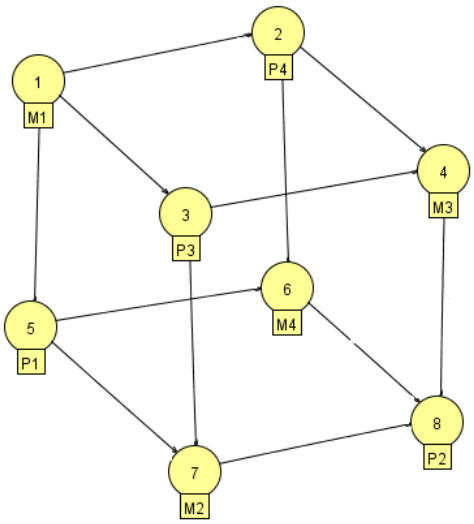


Figure 5.3 – Exemple de graphe à 8 nœuds

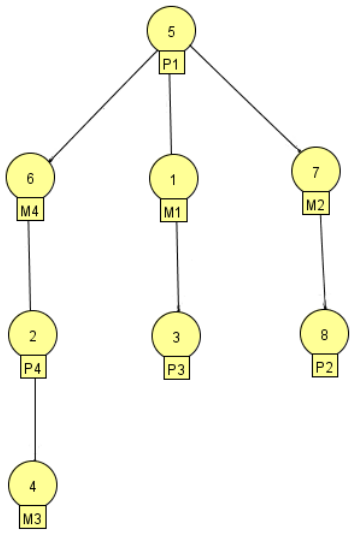


Figure 5.4 – Exemple d'arbre aléatoire associé à P_1

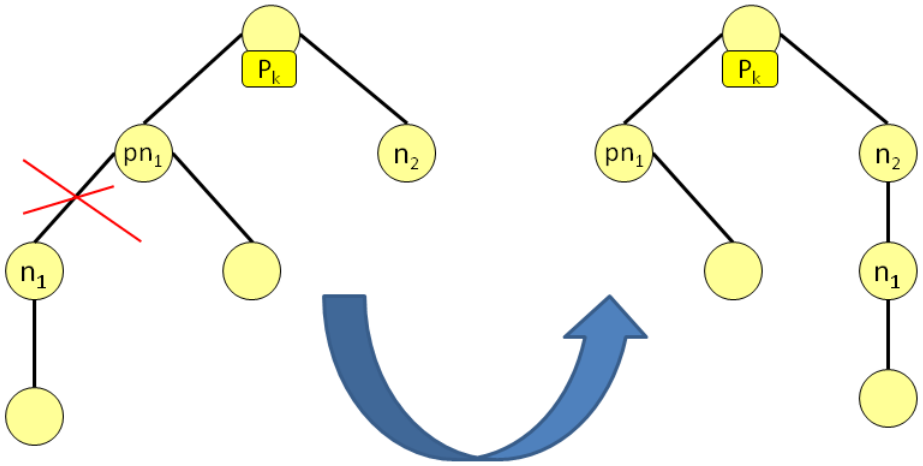


Figure 5.5 – Application d'un mouvement dans l'arbre P_k

L'architecture particulière de nos modèles permet d'avoir un nombre limité de mouvements puisque chaque nœud a un nombre limité de nœuds adjacents. En guise d'illustration, reprenons l'arbre associé à P_1 illustré à la Figure 5.4. La liste des mouvements pour cet arbre est donnée dans le Tableau 5.I.

Voisinage (Définition) *Un voisinage est un ensemble de mouvements qui peuvent être appliqués à une solution courante pour en donner une nouvelle.*

Arbre	n_1	pn_1	n_2
P_1	1	5	2
P_1	2	6	1
P_1	3	1	4
P_1	3	1	7
P_1	4	2	3
P_1	4	2	8
P_1	6	5	8
P_1	7	5	3
P_1	8	7	4
P_1	8	7	6

Tableau 5.I – Mouvements possibles pour l'arbre de la Figure 5.4

Par exemple, si nous considérons cet arbre associé à P_1, le voisinage correspond aux 10 mouvements applicables que nous retrouvons dans le Tableau 5.I. Dans notre implémentation de la méthode avec tabous, nous avons défini deux types de voisinage, un premier restreint (noté VR) et un second plus étendu (noté VE).

Le premier voisinage (VR) se résume ainsi :

1. Nous sélectionnons aléatoirement un indice k qui détermine l'arbre (associé à P_k) que nous allons modifier.
2. Nous faisons la liste de tous les mouvements possibles pour cet arbre.
3. Le voisinage correspond à l'ensemble des mouvements ainsi définis.

Le deuxième voisinage (VE) se veut plus étendu :

1. Pour tous les indices k, nous faisons la liste de tous les mouvements possibles pour tous les arbres associés à P_k.
2. Le voisinage correspond à l'ensemble des mouvements ainsi définis.

Nous déduisons aisément que le voisinage étendu est environ K fois plus grand que le voisinage restreint, si nous considérons que le nombre de mouvements possibles pour chaque arbre est environ le même. Dans le chapitre 6 nous comparerons numériquement les résultats obtenus avec chacun des voisinages.

5.3.3 Liste de tabous

À chaque itération, la meilleure solution du voisinage courant devient la solution courante de la prochaine itération. Cependant, cette solution peut ne pas améliorer la meilleure solution trouvée jusqu'ici, indiquant alors une forme d'optimum local. Pour éviter de retourner à un optimum local déjà visité et donc de former des cycles dans la recherche, nous gardons en mémoire les derniers mouvements effectués et nous interdisons de les utiliser pour un certain nombre d'itérations.

Plus formellement, en reprenant l'exemple de la Figure 5.5, supposons que notre nouvelle solution courante provient de l'application du mouvement (P_k, n_1, pn_1, n_2), c'est-à-dire en éliminant l'arête (pn_1, n_1). Pour éviter les cycles, nous interdisons l'arête (pn_1, n_1) d'être dans l'arbre P_k correspondant à une solution réalisable pendant s itérations, où s est la taille de la liste de tabous.

Notons que chacun des K arbres qui correspondent aux groupes de processeurs P_k, $k \in 1, \cdots, K$ possède sa propre liste de tabous. Chaque liste contient des couples de nœuds (pn, n). À chaque fois que nous rajoutons un couple en bout de liste, nous décalons tous les éléments d'une position, et donc le plus ancien couple de la liste n'en fait plus partie. Pour chaque mouvement testé, nous devons déterminer s'il est tabou en parcourant la liste de tabous.

Pour éviter cette recherche coûteuse au niveau de l'implémentation, nous utilisons une ruse : chaque liste de tabous est définie comme une matrice où l'élément (pn, n) correspond à la valeur de l'itération future à partir de laquelle le mouvement (pn, n) n'est plus tabou. Ainsi, pour tester si un mouvement est tabou, une seule comparaison suffit, et cela réduit grandement la complexité globale de la recherche.

Par ailleurs, il est important d'avoir à l'esprit l'influence de la taille de la liste de tabous sur l'efficacité la recherche. Si elle est trop petite, nous allons créer des cycles ; si elle est trop grande, nous allons interdire trop de mouvements intéressants et il faudra donc beaucoup plus d'itérations pour arriver à la solution désirée. Nous pouvons alors changer la taille de la liste de façon aléatoire tout au long de la recherche, c'est

ce que nous avons choisi de faire, conformément à l'étude faite par Taillard [58]. Nous détaillerons ce processus dans le chapitre 6 consacré aux résultats numériques.

5.3.4 Aspiration et favorisation

La liste de tabous est très utile, puisqu'elle permet d'éviter le phénomène de cycles, qui ferait boucler l'algorithme indéfiniment. Cependant, elle peut parfois empêcher l'application de certains mouvements intéressants, comme par exemple ceux qui mènent à une solution meilleure que la meilleure trouvée jusqu'à présent. C'est pour cette raison que le critère d'aspiration couramment utilisé est celui qui permet à un mouvement tabou d'être sélectionné s'il mène à une solution meilleure que la meilleure solution trouvée jusqu'à présent.

Cependant, nous remarquons que les problèmes que nous traitons font apparaître des matrices de flot qui contiennent beaucoup de valeurs nulles. Au fur et à mesure de la recherche avec tabous, des optima locaux sont atteints, suite à quoi les prochains mouvements choisis ne font qu'augmenter très peu la fonction objectif, ce qui ne permet pas *in fine* de sortir de cet optimum local. Ce phénomène provient du fait que plus de la moitié des éléments des matrices de flot utilisées sont nuls, par construction.

Pour faire face à cela, nous utiliserons un critère de favorisation proposé par Taillard [58]. Un mouvement est dit *favorisé* s'il n'est plus tabou depuis plus de *fav* itérations (ou bien s'il n'a jamais été tabou et n'a pas été appliqué depuis plus de *fav* itérations). Nous espérons ainsi diversifier la recherche et sortir des optima locaux. Nous reviendrons sur l'utilité de ce critère à la section 5.4 et nous détaillerons la valeur de *fav* dans le chapitre 6 consacré aux résultats numériques.

La solution initiale, le voisinage, la liste de tabous et les critères d'aspiration et de favorisation sont donc des notions clés du processus de recherche avec tabous, dont nous allons détailler le pseudocode complet.

5.4 Pseudocode de la recherche avec tabous

Dans cette section, nous allons dans un premier temps décrire le fonctionnement général de l'algorithme de recherche avec tabous que nous avons implémenté. Ensuite, nous décrirons le processus de choix d'un mouvement à l'aide d'arbres de décision. Enfin, nous présenterons le pseudocode de l'algorithme dans le cas du voisinage étendu, c'est-à-dire qu'à partir d'une solution courante, nous allons considérer dans le voisinage tous les mouvements possibles, pour tous les arbres correspondant à $P_k,\ k = 1, \cdots, K$.

5.4.1 Étapes essentielles

Nous pouvons distinguer sept grandes étapes d'une recherche avec tabous :

1. Générer une solution initiale et définir ainsi une première solution courante
2. Initialiser la liste de tabous, la solution courante et la meilleure solution
3. Définir le voisinage de la solution courante, c'est-à-dire lister l'ensemble des mouvements applicables
4. Choisir dans ce voisinage le meilleur mouvement conformément aux critères « tabou », « aspiration » et « favorisation ».
5. Appliquer le mouvement retenu
6. Mettre à jour la liste de tabous, la solution courante et la meilleure solution le cas échéant
7. Tant qu'aucun critère d'arrêt n'est satisfait, retourner à 3

5.4.2 Processus de sélection du mouvement le plus approprié

Définissons quelques notations qui permettent de comprendre la quatrième étape de la recherche avec tabous, dans laquelle nous devons sélectionner le mouvement le plus approprié parmi ceux qui définissent le voisinage de la solution courante. Pour ce faire,

considérons l'ensemble des mouvements du voisinage courant et définissons trois sous-ensembles comme suit :

- ASP : ensemble des mouvements *aspirés*
- FAV : ensemble des mouvements *favorisés*
- NONTAB : ensemble des mouvements non tabous

Remarquons qu'un mouvement donné peut appartenir à aucun, un, ou plusieurs de ces ensembles. D'autre part, nous avons la relation FAV $\subseteq$ NONTAB puisqu'un mouvement *favorisé* est non tabou, entre autres.

Pour expliquer le processus de sélection du mouvement le plus approprié, nous présentons deux arbres de décision qui décrivent la démarche de sélection dans le cas général, puis avec la solution que nous avons mise en œuvre.

Recherche avec tabous classique Le premier arbre illustre à la Figure 5.6 une recherche avec tabous classique, c'est-à-dire en utilisant exclusivement le critère d'aspiration.

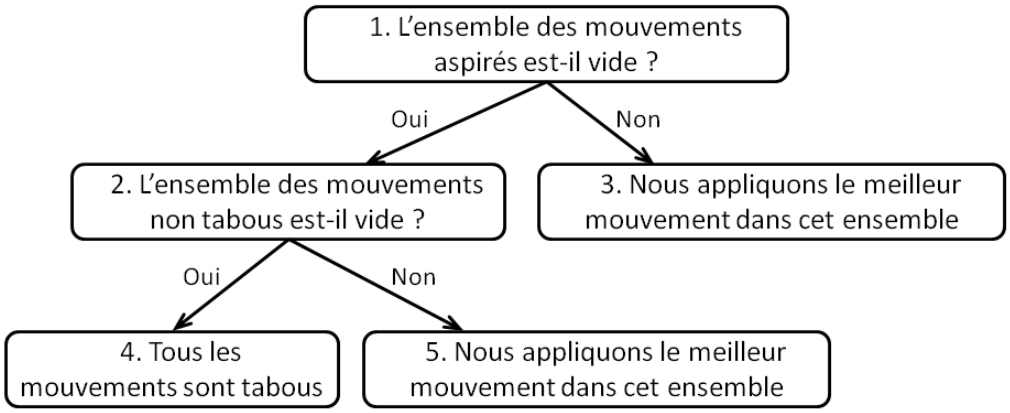


Figure 5.6 – Recherche avec tabous classique : sélection du mouvement à appliquer

Le processus essaie d'améliorer la meilleure solution actuelle en sélectionnant le meilleur mouvement *aspiré*, le cas échéant. Si aucun mouvement n'est *aspiré*, nous sé-

lectionnons le meilleur mouvement non tabou. Dans le cas extrême où tous les mouvements sont tabous, aucun mouvement n'est appliqué et nous passons à l'itération suivante.

Recherche avec tabous améliorée La recherche avec tabous que nous avons implémentée est plus subtile : nous allons nous servir du critère de favorisation pour diversifier davantage la recherche à long terme, comme illustré à la Figure 5.7.

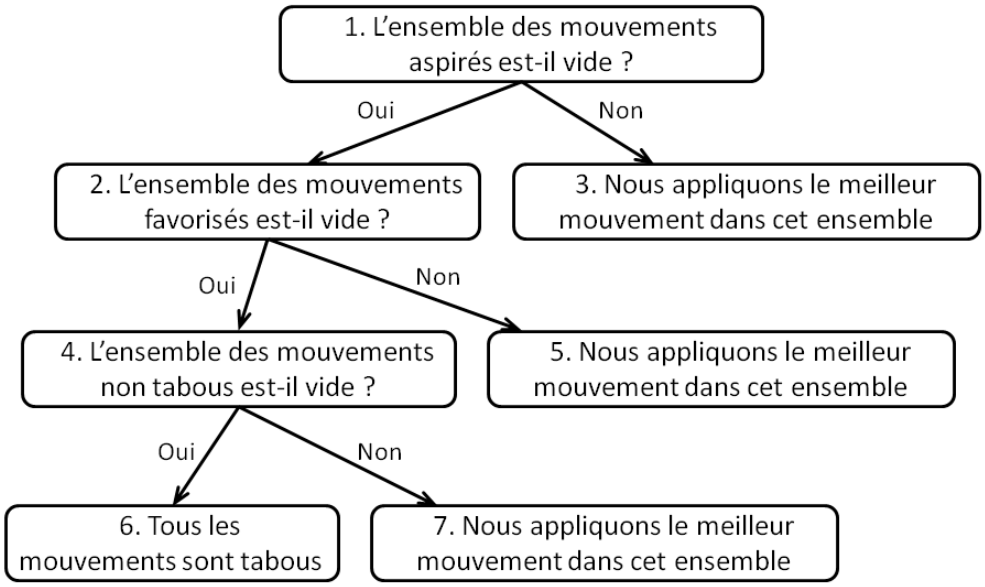


Figure 5.7 – Recherche avec tabous avec aspiration et favorisation : sélection du mouvement à appliquer

Tout commence de la même manière que pour une recherche classique. La différence vient du fait que si aucun mouvement n'est *aspiré*, alors nous sélectionnons le meilleur mouvement parmi un sous ensemble des mouvements non tabous. En effet, si l'ensemble des mouvements *favorisés* n'est pas vide, nous prenons le meilleur dans cet ensemble. La diversification provient du fait que l'espace de sélection est réduit par rapport à la recherche classique, puisque FAV $\subseteq$ NONTAB. La suite de l'arbre de décision est semblable à la recherche classique.

Grâce à cette astuce, nous arrivons à sortir complètement d'un optimum local pour explorer une autre région de solutions admissibles. C'est donc ce mode de sélection du mouvement à appliquer que nous avons implémenté, comme nous allons le voir dans le pseudocode de la recherche avec tabous.

5.4.3 Pseudocode correspondant à l'algorithme implémenté

À l'aide du processus de sélection du mouvement le plus approprié illustré à la Figure 5.7, nous pouvons écrire le pseudocode complet correspondant à la recherche avec tabous que nous avons implémentée.

Comme mentionné dans l'Algorithme 3, nous effectuons plusieurs résolutions dans une recherche avec tabous (nous travaillerons avec 20 résolutions dans le chapitre 6). Cela permet notamment de partir de plusieurs solutions initiales différentes et donc d'améliorer les résultats de la recherche. Notons qu'au cours d'une résolution donnée, si la meilleure solution n'est pas améliorée pendant *maxIterSansAmelioration* successives, alors nous arrêtons la résolution courante et nous passons à la suivante.

Précisons enfin que dans le code réellement implémenté, avec lequel nous avons effectué nos tests, nous avons optimisé la complexité de la recherche pour gagner en performances. En effet, il est possible de réaliser la sélection du mouvement le plus approprié en bouclant une seule fois sur l'ensemble des mouvements applicables, c'est-à-dire en parcourant une seule fois le voisinage.

Algorithme 3 : Algorithme de recherche avec tabous.

Sorties : *meilleur coût, meilleure solution*

pour *no res* $\leftarrow 1$ **à** *nb resolutions* **faire**
- Générer une solution initiale et calculer son coût
- Initialiser *solution courante, coût courant*
- Initialiser *meilleure solution, meilleur coût*
- Initialiser les listes de tabous
- **tant que** $nbIterSansAmelioration < maxIterSansAmelioration$ **faire**
  - Calculer le voisinage de la solution courante
  - Calculer les ensembles *ASP, FAV et NONTAB*
  - **si** *ASP est non vide* **alors**
    - mouvement à appliquer $\leftarrow \arg\min(ASP)$
  - **sinon si** *FAV est non vide* **alors**
    - mouvement à appliquer $\leftarrow \arg\min(FAV)$
  - **sinon si** *NONTAB est non vide* **alors**
    - mouvement à appliquer $\leftarrow \arg\min(NONTAB)$
  - **fin**
  - **si** *mouvement à appliquer existe* **alors**
    - mettre à jour *solution courante*
    - mettre à jour *coût courant*
    - mettre à jour la liste de tabous
    - **si** *coût courant* $<$ *meilleur coût* **alors**
      - *meilleur coût* $\leftarrow$ *coût courant*
      - *meilleure solution* $\leftarrow$ *solution courante*
    - **fin**
  - **fin**
- **fin**

fin

retourner *meilleur coût, meilleure solution*

La recherche avec tabous est performante grâce à la l'équilibre de ses composants principaux : type de voisinage, taille variable de la liste de tabous, critère d'aspiration et de favorisation. Nous verrons dans le chapite 6 que la bonne calibration de ces paramètres permet d'obtenir des résultats satisfaisants pour les différentes instances que nous avons voulu tester.

CHAPITRE 6

RÉSULTATS OBTENUS

Dans ce chapitre nous allons exposer les résultats numériques obtenus en utilisant les algorithmes décrits précédemment, à savoir la première et la deuxième affectation des composantes en utilisant un algorithme de colonies de fourmis et la résolution du problème de sélection des bus en utilisant une recherche avec tabous. Dans un souci de clarté, nous utiliserons les abréviations décrites dans le Tableau 6.I pour désigner ces trois problèmes.

QAP	Affectation sans prise en compte de l'importance thermique
QAP_{biaise}	Affectation avec prise en compte de l'importance thermique
BSP	Problème de sélection des bus

Tableau 6.I – Abréviations des trois différents problèmes résolus

Après la description des problèmes étudiés et de l'environnement de travail, nous ferons dans un premier temps une analyse sur la sélection des différents paramètres de l'algorithme de recherche avec tabous. La seconde partie du chapitre sera consacrée aux résultats obtenus, que nous commenterons.

6.1 Taille des problèmes étudiés

Nous avons étudié des problèmes de différentes tailles, comme indiqué dans le Tableau 6.II. Chaque problème est construit à partir d'une matrice d'interaction IA générée aléatoirement, sauf pour PR2 et PR4 où nous avons repris les matrices de la thèse d'Élie [20].

Un problème est décrit par une structure en trois dimensions et une matrice d'interaction IA. La structure en trois dimensions est composée d'un assemblage de blocs et est représentée par un graphe à n nœuds. La matrice IA précise les interactions entre les groupes de processeurs et les familles de mémoires. Par exemple, les entrées du problème PR2 sont illustrées aux Figures 6.1 et 6.2.

Nom du problème	Nombre de cubes	Nombre de nœuds
PR1	1	8
PR2	2	12
PR3	3	16
PR4	4	18
PR5	5	22
PR6	6	24
PR7	7	26

Tableau 6.II – Taille des problèmes étudiés

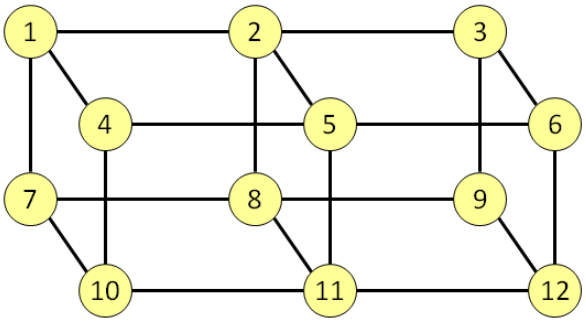


Figure 6.1 – Graphe associé au problème PR2

	M1	M2	M3	M4	M5	M6
P1	1843	25	20	18	35	216
P2	8	1509	50	37	42	36
P3	23	113	1838	36	29	47
P4	37	374	257	1763	22	0
P5	214	0	0	28	533	2
P6	6	0	1	153	91	1513

Figure 6.2 – Matrice d'interaction *IA* associée au problème PR2

Les matrices *IA* et les graphes associés aux problèmes PR1 à PR6 sont disponibles aux Annexes V et VI, respectivement.

6.2 Implémentation et environnement de travail

L'ensemble du code correspondant aux méthodes utilisées dans ce mémoire est écrit en langage Scilab 5.4.0 et testé sur une machine Windows ayant un processeur Intel Xeon W3520 cadencé à 2,67 GHz avec 4 GB de mémoire RAM.

6.3 Sélection des paramètres pour l'algorithme de colonies de fourmis

Nous utilisons l'algorithme de colonies de fourmis proposé par Taillard [56] pour résoudre le problème d'affectation quadratique. Son implémentation ne requiert que deux paramètres : le nombre d'itérations désirées et un paramètre R qui définit à quel point

nous voulons renforcer la trace de phéromones lors de la recherche.

Nous avons choisi de conserver les paramètres de l'implémentation de Taillard pour résoudre efficacement le QAP. Nous réaliserons donc 500 itérations pour chaque problème, et nous prendrons $R = 5$.

6.4 Sélection des paramètres pour l'algorithme de recherche avec tabous

Pour étudier l'influence des différents paramètres et calibrer au mieux l'algorithme, nous détaillons ici les résultats obtenus en se basant sur le problème PR4, en utilisant un voisinage étendu (VE), une solution initiale aléatoire, $\alpha = 0,5$ et en se basant sur l'affectation des composantes donnée par le problème d'affectation quadratique biaisé. Conformément à l'Algorithme 3 présenté à la section 5.4.3, nous devons calibrer trois paramètres pour réaliser notre recherche avec tabous :

- Le nombre maximal d'itérations sans amélioration, noté *maxIterSansAmelioration*. Au cours d'une résolution donnée, si la meilleure solution n'est pas améliorée pendant *maxIterSansAmelioration* successives, alors nous arrêtons la résolution courante et nous passons à la suivante.

- La taille moyenne de la liste de tabous, notée *tailleListeTabous*. Nous verrons que lorsqu'un mouvement est appliqué, il devient tabou pendant un certain nombre d'itérations. Ce nombre est variable puisqu'il est piloté par une formule faisant apparaitre un nombre aléatoire.

- Le critère de favorisation, noté *fav*. Il s'agit d'un entier, défini à la section 5.3.4, qui permet de calculer l'ensemble FAV, c'est-à-dire l'ensemble des mouvements qui ne sont plus tabous depuis plus de *fav* itérations.

6.4.1 Nombre maximal d'itérations sans amélioration

Nous avons vu précédemment que l'algorithme s'arrête lorsque la solution n'a pas été améliorée pendant *maxIterSansAmelioration* itérations successives. Pour fixer ce nombre

d'itérations, nous avons fixé les valeurs *tailleListeTabous* et *fav* conformément aux valeurs usuelles de la littérature [58]. L'étude suivante a donc été menée en fixant les deux derniers paramètres comme suit :

tailleListeTabous	30
fav	100

Le Tableau 6.III montre l'évolution du coût moyen final en fonction de *maxIterSansAmelioration*, pour 20 résolutions du problème de sélection des bus.

maxIterSansAmelioration	Coût moyen	Temps moyen (s)
0	710569	6,39
10	710703	9,97
20	730306	12,91
30	715528	18,43
40	705884	19,45
50	711912	22,31
60	697686	29,22
70	690486	35,28
80	687146	36,45
90	685202	38,91
100	696572	40,48
150	683330	51,18
200	685099	64,53
250	680572	78,32
300	681095	87,22

Tableau 6.III – Évolution du coût moyen et du temps en fonction du nombre maximal d'itérations sans amélioration avant arrêt de la résolution courante

Sur la Figure 6.3, nous pouvons noter qu'au delà de *maxIterSansAmelioration* = 90, la diminution de la valeur de la fonction objectif ne justifie pas d'augmenter le nombre maximal d'itérations sans amélioration compte tenu du temps de calcul nécessaire. En effet, il devient de plus en plus difficile d'améliorer la meilleure solution au fur et à mesure que le nombre d'itérations augmente.

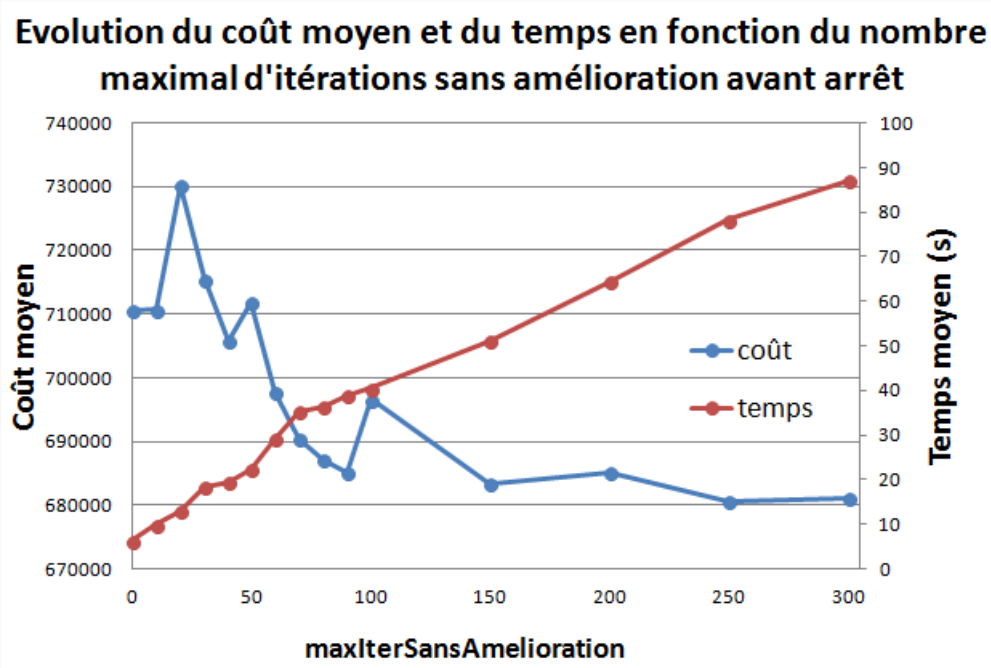


Figure 6.3 – Évolution du coût moyen et du temps en fonction du nombre maximal d'itérations sans amélioration avant arrêt de la résolution courante

Nous pouvons observer que le coût moyen diminue faiblement, voire stagne au delà de *maxIterSansAmelioration* = 150. Le temps de calcul, quant à lui, augment nécessairement lorsque le nombre d'itérations augmente. Nous garderons donc *maxIterSansAmelioration* = 90 comme étant une bonne valeur.

6.4.2 Taille moyenne de la liste de tabous

Nous avons choisi de mettre à jour la liste de tabous en interdisant l'arête retirée d'être à nouveau dans une solution pendant un certain nombre d'itérations. Pour éviter le phénomène de cycles et favoriser la diversification de l'algorithme dans le cas où la solution initiale n'est pas aléatoire, ce nombre d'itérations est un nombre aléatoire entier (noté durée taboue) dont nous voulons fixer la valeur moyenne. Déterminer la taille de la liste de tabous revient alors à déterminer le paramètre *tailleListeTabous* qui représente cette moyenne.

Notons que la taille moyenne de la liste de tabous n'est pas le nombre d'itérations

pour lesquelles le mouvement considéré sera tabou. En effet, la durée taboue est calculée comme suit :

$$\text{durée taboue} = rand \times tailleListeTabous$$

où *rand* est un nombre réel aléatoire entre 0 et 2.

Pour cette étude, nous fixons *maxIterSansAmelioration* avec la valeur trouvée précédemment et *fav* conformément à ce que propose la littérature pour des problèmes de cette taille [58] :

maxIterSansAmelioration	90
fav	100

Le Tableau 6.IV montre l'évolution du coût en fonction de la taille moyenne de la liste de tabous. Nous pouvons remarquer que si la taille moyenne de la liste de tabous est nulle, alors la solution trouvée est très médiocre.

tailleListeTabous	Coût
0	712915
5	691845
10	692641
15	691634
20	693650
25	685754
30	683562
35	688645
40	683370
45	697442
50	687507
55	692318
60	694499
65	688572
70	686986
75	690501

Tableau 6.IV – Évolution du coût en fonction de la taille moyenne de la liste de tabous

Nous pouvons expliquer cela par le fait que si aucun mouvement n'est tabou, alors le phénomène de cycle se produit inévitablement, d'où la solution de mauvaise qualité. En

effet, lors des premières itérations, la recherche avec tabous se comporte comme une méthode de descente classique. À partir du moment où le premier optimum local est atteint, nous allons dégrader la meilleure solution en appliquant un mouvement qui va faire augmenter la valeur de la fonction objectif. Ensuite, il se peut que nous retournions à l'optimum local déjà visité, puisqu'aucun mouvement n'est tabou ($tailleListeTabous = 0$).

Ce raisonnement est aussi valable avec une petite valeur du paramètre *tailleListeTabous*, seulement les cycles seront « plus longs », comme indiqué dans la première partie du graphe de la Figure 6.4, lorsque la taille moyenne de la liste de tabous est inférieure à 20.

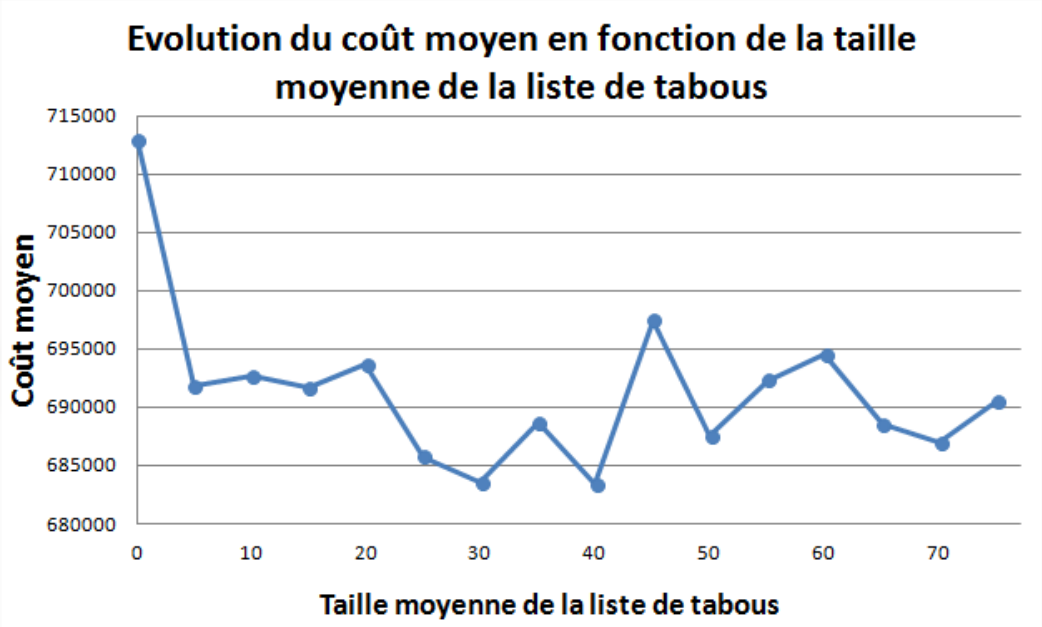


Figure 6.4 – Évolution du coût en fonction de la taille moyenne de la liste de tabous

C'est la raison pour laquelle, conformément au graphe de la Figure 6.4, nous choisirons *tailleListeTabous* = 30 comme étant une bonne valeur pour le problème étudié. Nous devons maintenant calibrer le dernier paramètre, à savoir le critère de favorisation.

6.4.3 « Favorisation »

Le critère d'aspiration couramment utilisé est celui qui permet à un mouvement tabou d'être sélectionné s'il mène à une solution meilleure que la meilleure trouvée jusqu'à présent. Dans un souci de diversification des différentes solutions parcourues, nous avons décidé d'utiliser un deuxième critère permettant notamment de sortir des optima locaux.

Nous rappelons qu'un mouvement est dit *favorisé* s'il n'est plus tabou depuis plus de *fav* itérations. Pour sélectionner ce critère, nous effectuons des tests en fixant les autres paramètres avec les valeurs trouvées précédemment :

maxIterSansAmelioration	90
tailleListeTabous	30

Le Tableau 6.V montre l'évolution du coût en fonction de *fav*. Le graphique correspondant est présenté à la Figure 6.5. Remarquons que le coût oscille à la hausse comme à la baisse en fonction de la valeur du critère *fav*.

fav	Coût
0	691164
10	691079
20	694208
30	683953
40	689528
50	690425
60	685665
70	685774
80	690954
90	688137
100	681349
150	694256
200	685320
250	699739

Tableau 6.V – Évolution du coût en fonction de la favorisation

Nous pouvons l'expliquer par le caractère aléatoire de la recherche (la taille de la liste de tabous varie) et le fait que le critère *fav* permet de diversifier la recherche. En

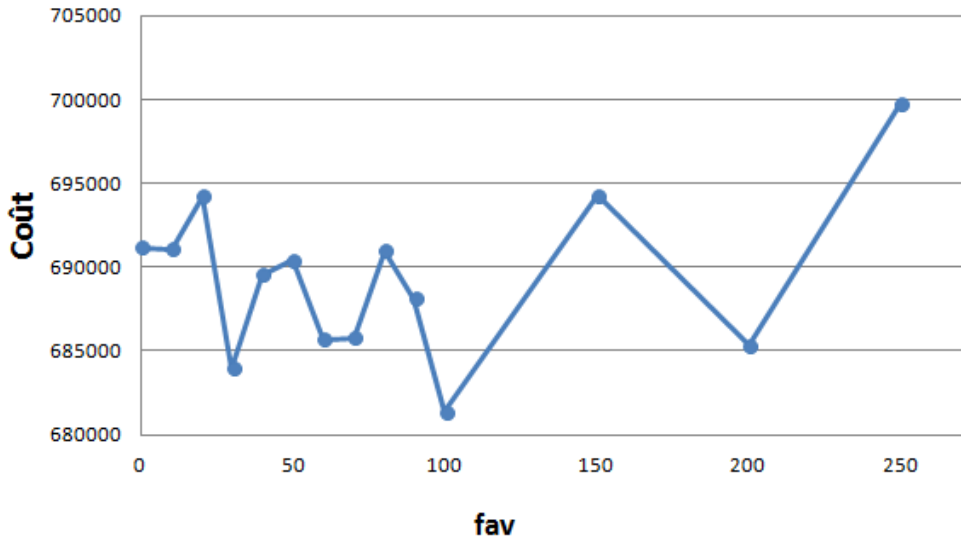


Figure 6.5 – Évolution du coût en fonction de la favorisation

effet, plus *fav* augmente, plus le cardinal du sous-ensemble FAV diminue et plus nous diversifions la recherche, comme nous l'illustrons dans le Tableau 6.VI.

Valeur de *fav*	Caractérisation du sous-ensemble FAV	Comportement de la recherche avec tabous
0	FAV = NONTAB	La recherche se comporte comme si nous n'avions pas le critère de favorisation
moyenne	FAV $\subset$ NONTAB	Il y a diversification de la recherche avec tabous
très grande	FAV $= \emptyset$	La recherche se comporte comme si nous n'avions pas le critère de favorisation

Tableau 6.VI – Caractérisation du sous-ensemble FAV en fonction de la valeur de *fav*

De façon plus précise, lorsque $fav = 0$, nous avons FAV = NONTAB et la recherche avec tabous agit donc comme si nous n'avions pas introduit l'ensemble FAV. Lorsque *fav* a une valeur moyenne, alors nous avons FAV $\subset$ NONTAB et nous diversifions la recherche en choisissant le meilleur mouvement dans l'ensemble FAV, comme indiqué à la Figure 5.7 de la section 5.4.2. Enfin, lorsque *fav* est très grand, alors il y a de fortes

chances pour que l'ensemble FAV soit vide, auquel cas nous appliquons le meilleur mouvement non tabou.

Sur le graphique de la Figure 6.5, nous pouvons observer que le meilleur coût moyen est obtenu avec *fav* = 100, c'est donc cette valeur que nous allons garder.

6.4.4 Paramètres fixés pour tous les problèmes

Nous avons effectué l'étude précédente sur les sept problèmes PR1 à PR7 avec voisinage étendu et restreint. Les résultats de la sélection des meilleurs paramètres sont exposés dans les Tableaux 6.VII et 6.VIII.

	PR1	PR2	PR3	PR4	PR5	PR6	PR7
Nombre de mouvements	≈ 31	≈ 85	≈ 165	≈ 240	≈ 370	≈ 457	≈ 558
maxIterSansAmelioration	12	45	60	90	160	220	290
tailleListeTabous	8	14	18	30	31	39	40
fav	13	40	90	100	140	210	250

Tableau 6.VII – Paramètres obtenus pour les différents problèmes avec un voisinage étendu (VE)

	PR1	PR2	PR3	PR4	PR5	PR6	PR7
Nombre de mouvements	≈ 8	≈ 14	≈ 20	≈ 27	≈ 33	≈ 38	≈ 43
maxIterSansAmelioration	3	7	7	10	13	18	22
tailleListeTabous	5	5	6	9	11	11	11
fav	4	8	12	15	16	17	25

Tableau 6.VIII – Paramètres obtenus pour les différents problèmes avec un voisinage restreint (VR)

Le nombre de mouvements pour un problème est approximatif car pour chaque mouvement appliqué, nous obtenons un nouvel arbre auquel est associé un nouveau voisinage, c'est-à-dire un certain nombre de mouvements applicables. Nous remarquons que ce nombre de mouvements varie avec une amplitude faible. Le nombre de mouvements indiqué dans les Tableaux 6.VII et 6.VIII est en fait le nombre moyen de mouvements pour chaque problème.

Nous avons représenté graphiquement les résultats associés au voisinage étendu sur les graphes des Figures 6.6 et 6.7 en indiquant à chaque fois une courbe décrivant une fonction connue se rapprochant des courbes obtenues lors de nos tests. C'est ainsi que nous allons déduire la formule qui permet d'initialiser les paramètres pendant une recherche avec tabous.

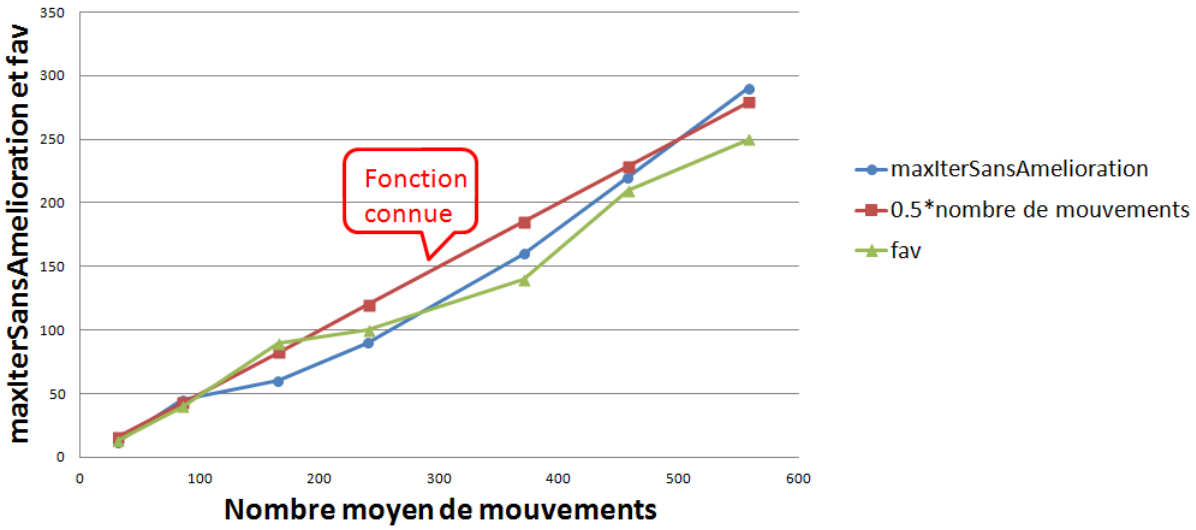


Figure 6.6 – Évolution des paramètres maxIterSansAmelioration et fav en fonction du nombre moyen de mouvements dans le cadre d'un voisinage étendu (VE)

En ce qui concerne le graphique de la Figure 6.6, nous remarquons que l'évolution des critères *maxIterSansAmelioration* et *fav* est sensiblement identique, c'est pourquoi nous les représentons sur le même graphique. Nous trouvons alors qu'une fonction connue représentant les courbes est $0.5 \times$ nombre de mouvements.

Pour ce qui est de la taille moyenne de la liste de tabous, dont le graphique est présenté à la Figure 6.7, nous choisirons la fonction $\sqrt{3 \times \text{nombre de mouvements}}$. Notons que ces fonctions sont déduites des graphiques correspondant à un voisinage étendu. Nos études montrent que les valeurs trouvées pour le voisinage restreint acceptent aussi les mêmes fonctions. Les paramètres fixés pour la suite seront donc :

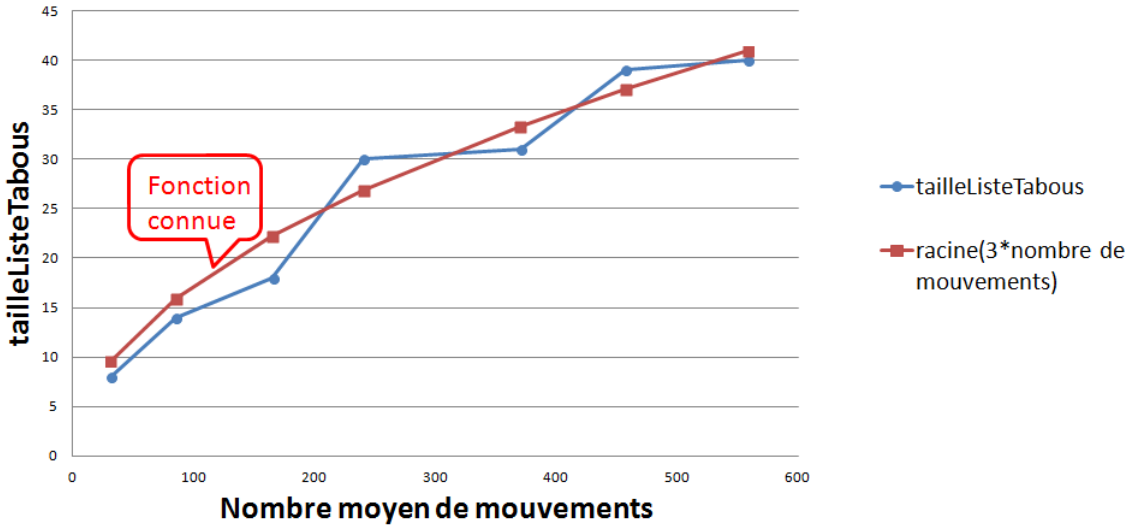


Figure 6.7 – Évolution du paramètre tailleListeTabous en fonction du nombre moyen de mouvements dans le cadre d'un voisinage étendu (VE)

tailleListeTabous	$\sqrt{3 \times \text{nombre de mouvements}}$
fav	0.5 x nombre de mouvements
maxIterSansAmelioration	0.5 x nombre de mouvements

Maintenant que les paramètres sont définis, nous allons préciser la nature des bus puis exposer les résultats obtenus avec des problèmes de différentes tailles. Notons que les paramètres utilisés pour l'expérimentation seront ceux déterminés par les formules ci-dessus (plutôt que ceux obtenus explicitement dans les Tableaux 6.VII et 6.VIII).

Au niveau de l'implémentation, nous avons choisi de calculer dynamiquement le nombre moyen de mouvements. À chaque itération, nous parcourons l'ensemble des mouvements, donc nous en connaissons le nombre. Tout au long de la recherche avec tabous, nous pouvons mettre à jour le nombre moyen de mouvements, et c'est cette valeur qui permet de calculer (et de mettre à jour) les trois paramètres définis ci-dessus. Rappelons encore une fois que le nombre de mouvements ne varie qu'avec une faible amplitude, ce qui assure d'utiliser des paramètres donc les variations restent raisonnables.

6.5 Création des bus

Nous allons créer l'ensemble des bus disponibles en tenant compte des informations suivantes : chaque bus w a une capacité u_w et un coût d'utilisation ς_{ew} sur l'arête $e \in E$. Puisque nous n'avions pas non plus d'information au niveau des capacités et des coûts des bus disponibles dans l'industrie, nous avons utilisé l'approche suivante pour les spécifier afin de compléter l'évaluation de nos procédures de résolution. Dans un souci d'équilibrer au mieux les flots dans la structure, le coût d'un bus de capacité u doit être supérieur au coût de deux bus de capacités $u/2$. Nous avons donc défini les bus comme indiqué dans le Tableau 6.IX. La formule de calcul du coût en fonction de la

Capacité	0	100	200	300	400	500	$\cdots$	4000
Coût	0	1500	4000	6500	9000	11500	$\cdots$	99000

Tableau 6.IX – Capacité et coût d'utilisation des bus disponibles

capacité est la suivante :

$$cout = 25capacite - 1000$$

Maintenant que les bus sont définis, nous pouvons exposer les différents résultats numériques obtenus.

6.6 Résultats numériques obtenus pour les différents problèmes

Dans cette section nous exposons les résultats obtenus pour les différents problèmes, en prenant les valeurs des paramètres obtenues par les formules définies à la fin de la section 6.4.4. Au niveau du voisinage (restreint ou étendu) et de la solution initiale du BSP (solution du problème d'affectation quadratique ou solution aléatoire), nous utilisons les notations suivantes :

- VR : Voisinage Restreint
- VE : Voisinage Étendu
- SIA : Solution Initiale Aléatoire, c'est-à-dire générée par l'algorithme de Wilson

- SIQAP : Initialisation avec la solution du problème d'affectation quadratique (biaisé ou non selon le cas), c'est-à-dire que nous utilisons les résultats de l'algorithme de Floyd-Warshall pour créer les arbres initiaux

6.6.1 Présentation des résultats

Nous présentons deux tableaux de résultats : le premier est obtenu à partir de l'affectation des composantes aux nœuds de la structure donnée par le problème *QAP*, le deuxième provient de l'affectation donnée par le problème QAP_{biaise}, c'est-à-dire en utilisant la matrice de distance biaisée.

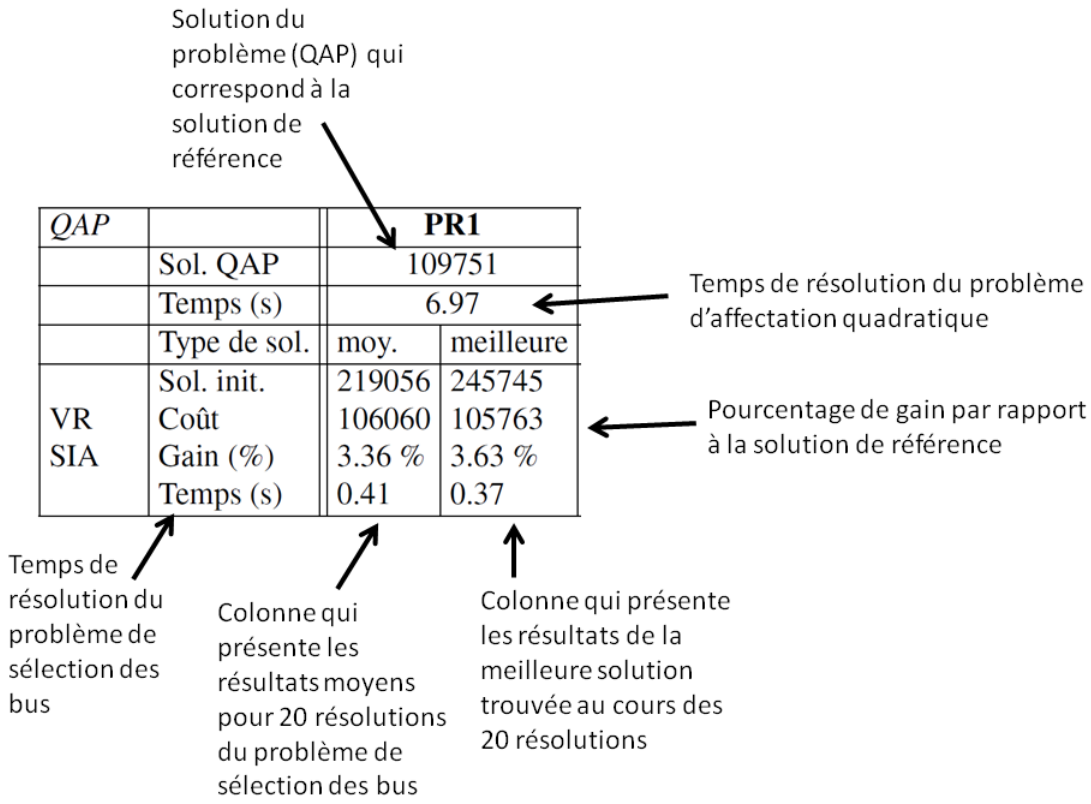
Nous dénotons ces deux affectations des composantes comme étant respectivement l'affectation originale (AFFOR) et l'affectation biaisée (AFFBI). Chaque tableau présente les quatre cas de figure possibles :

1. Voisinage Restreint avec Solution Initiale Aléatoire (VR & SIA)
2. Voisinage Restreint avec Solution Initiale = Solution du QAP (VR & SIQAP)
3. Voisinage Étendu avec Solution Initiale Aléatoire (VE & SIA)
4. Voisinage Étendu avec Solution Initiale = Solution du QAP (VE & SIQAP)

Les résultats numériques qui suivent ont été réalisés avec $\alpha = 0,5$. Le Tableau 6.X présente les résultats obtenus lorsque l'affectation AFFOR des composantes aux nœuds de la structure est celle donnée par le problème *QAP*. Le Tableau 6.XI provient de l'affectation AFFBI donnée par le problème QAP_{biaise}. Dans les deux tableaux, nous indiquons notamment l'évolution du coût par rapport à la solution donnée par le QAP (ou QAP_{biaise}) qui sert de référence. Cette évolution est exprimée en pourcentage. Les temps de calculs sont exprimés en secondes et nous distinguons dans les tableaux le temps associé à la résolution du problème d'affectation quadratique avec celui associé à la résolution du problème de sélection des bus.

Les valeurs des paramètres sont obtenues par les formules que nous avons déduites dans la section 6.4.4. Chaque problème a été résolu 20 fois et nous rappelons que dans la

cadre d'une solution initiale aléatoire (SIA), nous travaillons avec 20 solutions initiales différentes. Lors des résolutions où la solution initiale est celle obtenue à la suite du problème d'affectation quadratique, la solution initiale est toujours la même. La Figure 6.8 permet de mieux comprendre la lecture des tableaux de résultats.



QAP		**PR1**	
	Sol. QAP	109751	
	Temps (s)	6.97	
	Type de sol.	moy.	meilleure
VR SIA	Sol. init.	219056	245745
	Coût	106060	105763
	Gain (%)	3.36 %	3.63 %
	Temps (s)	0.41	0.37

Figure 6.8 – Lecture des tableaux de résultats

Au niveau du pourcentage de gain par rapport à la solution de référence, la formule de calcule est la suivante :

$$\text{Gain} = \frac{\text{Coût Sol. QAP} - \text{Coût}}{\text{Coût Sol. QAP}} \times 100$$

QAP		**PR1**		**PR2**		**PR3**		**PR4**		**PR5**		**PR6**		**PR7**	
	Sol. QAP	109751		692675		793014		728397		1491631		1644135		1830298	
	Temps (s)	6.97		18.15		35.5		47.73		78.50		105.00		119.50	
	Type de sol.	moy.	meilleure	moy.	meilleure	moy.	meilleure	moy.	meilleure	moy.	meilleure	moy.	meilleure	moy.	meilleure
	Sol. init.	219056	245745	2047466	1900517	1809451	1542963	2032927	2531221	3602125	3130581	3800359	3491677	4717541	5363156
VR	Coût	106060	105763	653355	650376	723773	713614	689428	679059	1390498	1371032	1361673	1342927	1569036	1547022
SIA	Gain (%)	3.36 %	3.63 %	5.68 %	6.11 %	8.73 %	10.01 %	5.35 %	6.78 %	14.78 %	15.98 %	17.18 %	18.32 %	14.27 %	15.47 %
	Temps (s)	0.41	0.37	2.36	4.45	6.71	6.08	12.18	19.04	27.64	32.19	38.88	31.68	56.99	73.48
VR	Coût	106060	105763	652638	650376	715610	710884	685625	680563	1379288	1368710	1342112	1330522	1536518	1528894
SIQAP	Gain (%)	3.36 %	3.63 %	5.78 %	6.11 %	9.76 %	10.36 %	5.87 %	6.57 %	15.47 %	16.12 %	18.37 %	19.07 %	16.05 %	16.46 %
	Temps (s)	0.33	0.52	2.21	2.29	6.52	9.05	12.27	12.44	20.08	36.36	41.06	49.63	55.49	64.28
	Sol. init.	240641	243225	1657615	778296	1672225	1345745	1898075	2449829	3438682	3579624	3792316	4396093	4369584	4087480
VE	Coût	107453	105763	656167	650376	723772	710884	691294	680867	1410931	1385233	1392657	1362986	1578432	1547574
SIA	Gain (%)	2.09 %	3.63 %	5.27 %	6.11 %	8.73 %	10.36 %	5.10 %	6.53 %	13.53 %	15.11 %	15.29 %	17.10 %	13.76 %	15.44 %
	Temps (s)	0.35	0.60	3.03	3.02	12.26	20.54	25.59	23.78	64.09	50.76	99.00	99.97	151.05	156.58
VE	Coût	107251	107251	651608	650376	712285	710892	687431	682029	1374292	1368573	1336771	1328862	1537076	1529728
SIQAP	Gain (%)	2.27 %	2.27 %	5.93 %	6.11 %	10.18 %	10.36 %	5.63 %	6.37 %	15.78 %	16.13 %	18.69 %	19.17 %	16.02 %	16.42 %
	Temps (s)	0.49	0.53	3.57	4.39	11.60	16.42	21.11	33.95	66.41	68.35	99.72	111.72	148.43	156.54

Tableau 6.X – Résultats numériques. Affectation des composantes AFFOR

QAP_{biaise}		**PR1**		**PR2**		**PR3**		**PR4**		**PR5**		**PR6**		**PR7**	
	Sol. QAP_{biaise}	107251		583728		647596		679897		1396584		1349688		1512637	
	Temps (s)	6.99		18.10		36.27		45.53		80.46		102.60		123.13	
	Type de sol.	moy.	meil.	moy.	meil.	moy.	meil.	moy.	meil.	moy.	meil.	moy.	meil.	moy.	meil.
	Sol. init.	219526	136507	1857748	2488378	1797565	1389866	2340181	2585278	3521777	3835768	4014802	3242124	4344750	4502135
VR	Coût	106507	105763	555534	553007	648697	632226	688736	665686	1382199	1368447	1351848	1325080	1504558	1470309
SIA	Gain (%)	0.69 %	1.38 %	4.83 %	5.26 %	-0.17 %	2.36 %	-1.30 %	2.08 %	1.03 %	2.01 %	-0.16 %	1.82 %	0.53 %	2.79 %
	Temps (s)	0.39	0.23	2.09	2.39	6.74	6.61	13.87	11.44	26.00	31.18	37.32	65.90	56.74	70.85
VR	Coût	107251	107251	557514	552982	635704	632204	668254	665778	1363815	1363811	1309842	1307838	1467778	1464957
$SIQAP_{biaise}$	Gain (%)	0 %	0 %	4.49 %	5.27 %	1.82 %	2.36 %	1.70 %	2.07 %	2.34 %	2.34 %	2.95 %	3.09 %	2.96 %	3.15 %
	Temps (s)	0.39	0.52	1.62	3.20	4.35	4.35	8.37	9.89	15.70	15.01	23.50	35.48	56.12	65.47
	Sol. init.	193656	267605	1799756	1334492	1943696	1687356	2107078	1808207	3658039	4486276	3887226	4054777	4629541	4848113
VE	Coût	107554	105763	558208	553007	656989	640133	683468	669457	1433324	1388620	1360508	1322308	1531532	1488267
SIA	Gain (%)	-0.28 %	1.38 %	4.37 %	5.26 %	-1.46 %	1.14 %	-0.52 %	1.53 %	-2.63 %	0.57 %	-0.80 %	2.02 %	-1.24 %	1.61 %
	Temps (s)	0.42	0.29	3.40	2.60	12.75	18.32	28.23	26.45	66.02	102.83	104.20	109.66	151.85	152.64
VE	Coût	107251	107251	555339	552982	633657	633657	665951	665012	1360801	1358180	1301032	1299827	1460251	1457349
$SIQAP_{biaise}$	Gain (%)	0 %	0 %	4.86 %	5.27 %	2.14 %	2.14 %	2.04 %	2.18 %	2.56 %	2.74 %	3.60 %	3.69 %	3.46 %	3.65 %
	Temps (s)	0.39	0.36	3.63	4.37	9.03	9.03	23.89	30.71	60.50	63.47	71.57	76.31	150.23	158.76

Tableau 6.XI – Résultats numériques. Affectation des composantes AFFBI

Les Figures 6.9 et 6.10 présentent les coûts moyens des différents problèmes en fonction du type de voisinage et de solution initiale, lorsque l'affectation des composantes est donnée par les problèmes *QAP* et QAP_{biaise}, respectivement.

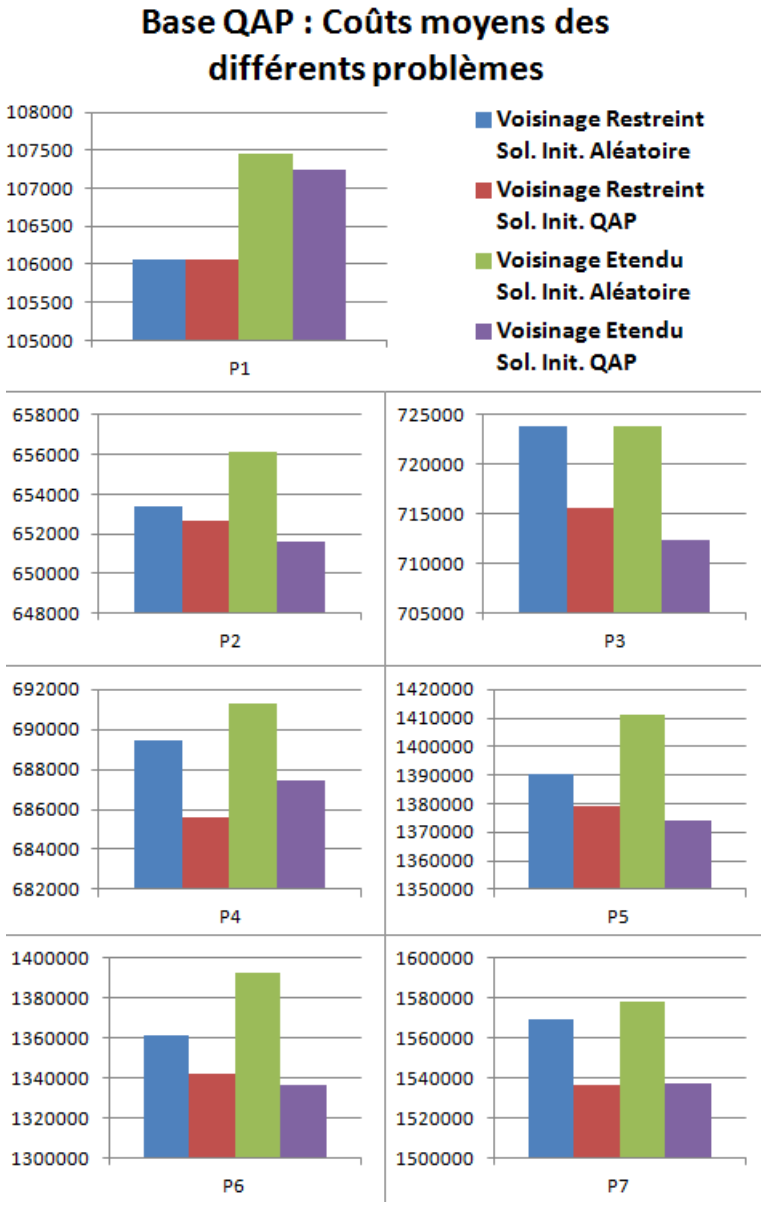


Figure 6.9 – Coûts moyens des différents problèmes avec affectation AFFOR

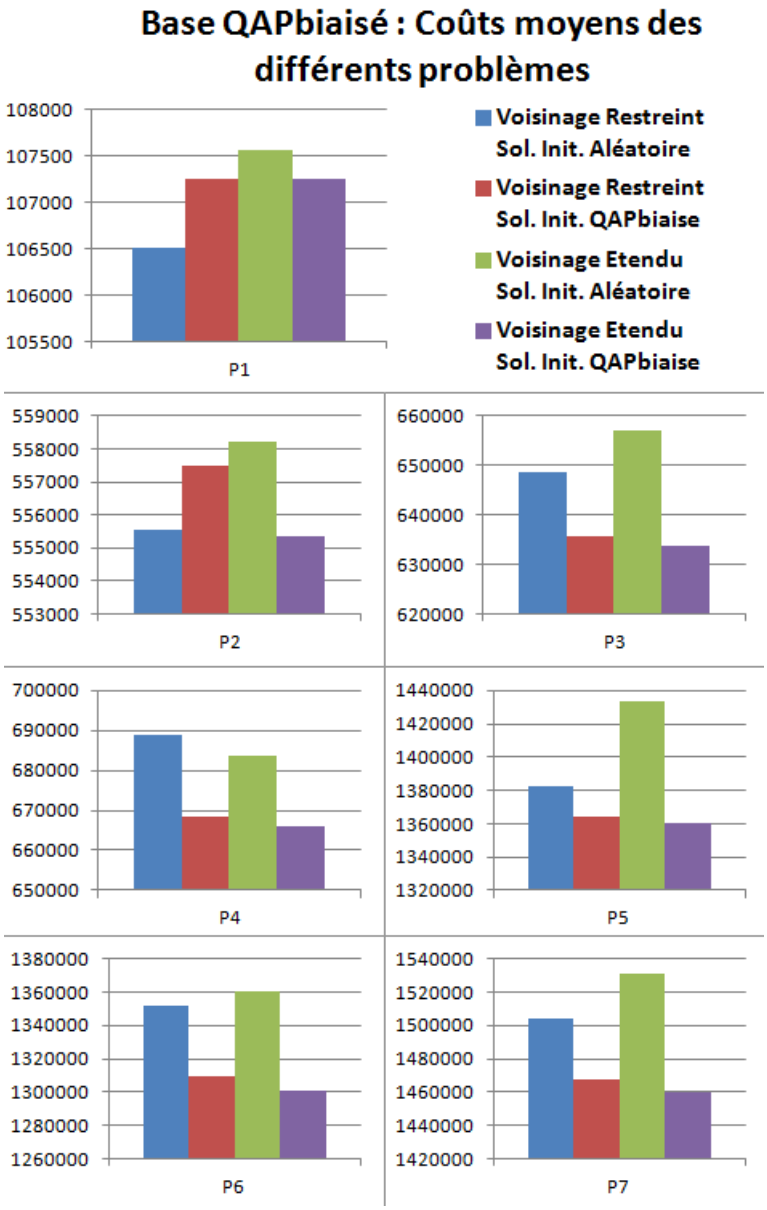


Figure 6.10 – Coûts moyens des différents problèmes avec affectation AFFBI

Dans les sections suivantes nous allons analyser différents éléments de ces résultats : l'effet des deux affectations AFFOR et AFFBI, le choix de la solution initiale (SIA versus SIQAP ou SIQAPbiaisé), le choix du voisinage (étendu ou restreint), et les temps de calcul.

6.6.2 Affectation des composantes AFFOR versus AFFBI

Comme nous l'avons vu dans le chapitre 4, nous résolvons un problème d'affectation quadratique pour affecter les groupes de processeurs et les familles de mémoires aux nœuds de l'architecture. Pour cela, nous pouvons utiliser la matrice de distance de base entre les nœuds pour obtenir l'affectation AFFOR ou bien nous pouvons décider de biaiser cette matrice de distance pour anticiper le coût futur des flots qui transiteront au centre de la structure pour obtenir l'affectation AFFBI. Il est intéressant de comparer les résultats du problème de sélection des bus, suivant ces deux affectations.

Pour cela, nous avons repris une partie des résultats des Tableaux 6.X et 6.XI, à savoir les coûts moyens obtenus pour les différents problèmes. Dans le Tableau 6.XII, nous indiquons notamment dans la troisième ligne le pourcentage de gain entre le coût obtenu avec AFFOR et le coût obtenu avec AFFBI.

		PR1	**PR2**	**PR3**	**PR4**	**PR5**	**PR6**	**PR7**
VR	AFFOR	106060	653355	723773	689428	1390498	1361673	1569036
SIA	AFFBI	106507	555534	648697	688736	1382199	1351848	1504558
	Gain (%)	-0,42 %	14,97 %	10,37 %	0,10 %	0,60 %	0,72%	4,11%
VR	AFFOR	106060	652638	715610	685625	1379288	1342112	1536518
SIQAP	AFFBI	107251	557514	635704	668254	1363815	1309842	1467778
	Gain (%)	-1,12%	14,58 %	11,17 %	2,53 %	1,12%	2,40%	4,47%
VE	AFFOR	107453	656167	723772	691294	1410931	1392657	1578432
SIA	AFFBI	107554	558208	656989	683468	1433324	1360508	1531532
	Gain (%)	-0,09%	14,93%	9,23%	1,13%	-1,59%	2,31%	2,97%
VE	AFFOR	107251	651608	712285	687431	1374292	1336771	1537076
SIQAP	AFFBI	107251	555339	633657	665951	1360801	1301032	1460251
	Gain (%)	0,00 %	14,77%	11,04%	3,12 %	0,98%	2,67%	5,00%

Tableau 6.XII – AFFOR versus AFFBI : coûts moyens pour les différents problèmes

Les résultats indiquent que les coûts obtenus avec l'affectation AFFBI sont meilleurs que ceux avec AFFOR. En effet, ceci est vérifié pour tous les problèmes, sauf PR1 et PR5, avec VE et SIA. Les solutions obtenues avec AFFOR sont améliorées de 5.5 % en moyenne. La meilleure amélioration est obtenue pour le problème PR2 (près de 15% quels que soient le voisinage et la solution initiale).

Nous pouvons expliquer ce phénomène de la façon suivante : l'affectation des composantes AFFBI tient déjà compte du dégagement de chaleur, puisque la matrice de

distance est biaisée. En conséquence, nous avons en quelque sorte anticipé le problème de sélection des bus en tenant compte du dégagement de chaleur.

Cela dit, sur certains problèmes, l'affectation des composantes AFFBI n'apporte pas nécessairement une amélioration conséquente, comme en témoigne le problème PR5. En effet, dans le cas de l'affectation AFFOR où les composantes qui interagissent beaucoup sont au centre de la structure, il est par exemple possible de trouver un chemin reliant deux composantes centrales, tout en tenant compte de la contrainte de dégagement de chaleur, comme nous l'indiquons en rouge à la Figure 6.11.

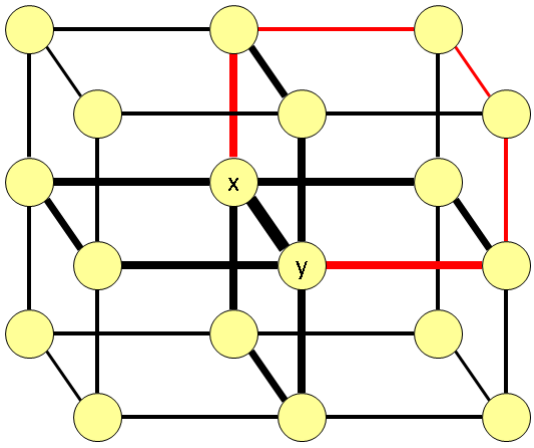


Figure 6.11 – Chemin possible pour relier deux composantes centrales

Évidemment, à partir de la composante x ou y, la première arête traversée par le flot est obligatoirement une arête centrale (plus épaisse sur la Figure 6.11). Même s'il est possible de trouver un chemin assez peu coûteux au niveau thermique, nous sommes tout de même obligés de sortir du centre de la structure par une ou plusieurs arêtes centrales.

6.6.3 Solution initiale aléatoire versus solution initiale donnée par le problème d'affectation quadratique

La solution initiale pour le problème BSP peut être la solution obtenue en dirigeant les flots entre les composantes le long des chemins identifiés avec l'algorithme de Floyd-Warshall lors de la résolution du problème d'affectation quadratique ou bien nous pouvons générer une solution initiale aléatoire, en utilisant l'algorithme de Wilson.

Sur les Figures 6.9 et 6.10, les bâtons bleus et verts indiquent les coûts moyens obtenus avec une solution initiale aléatoire alors que les bâtons rouges et violets correspondent à la solution initiale donnée lors de la résolution du problème (QAP).

Dans la plupart des cas, la solution initiale obtenue en dirigeant les flots entre les composantes le long des chemins identifiés avec l'algorithme de Floyd-Warshall engendre de meilleurs résultats. Rappelons que le problème BSP tente d'améliorer la solution justement obtenue après la résolution du problème d'affectation quadratique en redirigeant les flots sur les arêtes de la structure, tout en tenant compte des bus disponibles sur chacune des arêtes.

La solution initiale obtenue après application du problème d'affectation quadratique est donc d'une certaine qualité, comparée à l'utilisation d'une solution initiale complètement aléatoire dont le coût peut être aberrant. Ceci confirme que même si la recherche avec tabous permet de partir avec plusieurs solutions initiales aléatoires différentes pour diversifier la recherche, il est vrai que démarrer la recherche avec une bonne solution initiale est tout de même un avantage.

6.6.4 Voisinage étendu versus voisinage restreint

Rappelons que nous considérons deux voisinages distincts dont nous voulons comparer les performances. Le voisinage restreint (VR) correspond à l'ensemble des mouvements possibles pour un arbre donné, alors que le voisinage étendu (VE) est composé de tous les mouvements possibles pour tous les arbres.

De la même manière qu'à la section 6.6.2, nous avons repris les coûts moyens obtenus pour les différents problèmes dans les Tableaux 6.X et 6.XI. Dans le Tableau 6.XIII, nous nous intéressons au pourcentage de gain entre le coût obtenu avec le voisinage étendu (VE) et celui obtenu avec le voisinage restreint (VR) :

$$\text{Gain} = \frac{\text{coût avec VE} - \text{coût avec VR}}{\text{coût avec VE}} \times 100$$

Lorsque le voisinage restreint donne de meilleurs résultats que le voisinage étendu, le

pourcentage associé est indiqué en gras.

		PR1	**PR2**	**PR3**	**PR4**	**PR5**	**PR6**	**PR7**
SIA	Coût VE	107453	656167	723772	691294	1410931	1392657	1578432
AFFOR	Coût VR	106060	653355	723773	689428	1390498	1361673	1569036
	Gain (%)	**1,30%**	**0,43%**	0,00%	**0,27%**	**1,45%**	**2,22%**	**0,60%**
SIA	Coût VE	107554	558208	656989	683468	1433324	1360508	1531532
AFFBI	Coût VR	106507	555534	648697	688736	1382199	1351848	1504558
	Gain (%)	**0,97%**	**0,48%**	**1,26%**	-0,77%	**3,57%**	**0,64%**	**1,76%**
SIQAP	Coût VE	107251	651608	712285	687431	1374292	1336771	1537076
AFFOR	Coût VR	106060	652638	715610	685625	1379288	1342112	1536518
	Gain (%)	**1,11%**	-0,16%	-0,47%	**0,26%**	-0,36%	-0,40%	**0,04%**
SIQAP	Coût VE	107251	555339	633657	665951	1360801	1301032	1460251
AFFBI	Coût VR	107251	557514	635704	668254	1363815	1309842	1467778
	Gain (%)	0,00%	-0,39%	-0,32%	-0,35%	-0,22%	-0,68%	-0,52%

Tableau 6.XIII – Voisinage étendu versus voisinage restreint : coûts moyens pour les différents problèmes

Nous pouvons observer dans le Tableau 6.XIII que lorsque la solution initiale est aléatoire, c'est le voisinage restreint qui donne les meilleurs résultats. Par contre, lorsque la solution initiale est celle donnée par le problème d'affectation quadratique, alors les résultats sont plus mitigés et dépendent de la structure des problèmes.

Quand nous utilisons un voisinage étendu, le nombre de mouvements applicables est élevé et l'espace de recherche est large : nous diversifions ainsi la recherche. Au contraire, l'utilisation du voisinage restreint favorise l'exploration d'une partie de l'espace des solutions, nous intensifions alors les recherches localement lors de l'optimisation d'un arbre donné.

6.6.5 Temps de calcul de la recherche avec tabous

Pour illustrer l'évolution des temps de calcul de la recherche avec tabous, nous présentons aux Figures 6.12 et 6.13 l'évolution du temps de calcul en fonction du nombre de nœuds, pour le voisinage étendu et le voisinage restreint, respectivement.

Nous pouvons observer que le temps de calcul augmente avec le nombre de nœuds. En effet, plus le nombre de nœuds augmente, plus le nombre de mouvements possibles dans les différents arbres associés aux groupes de processeurs $P_1, P_2, \cdots, P_k$ augmente.

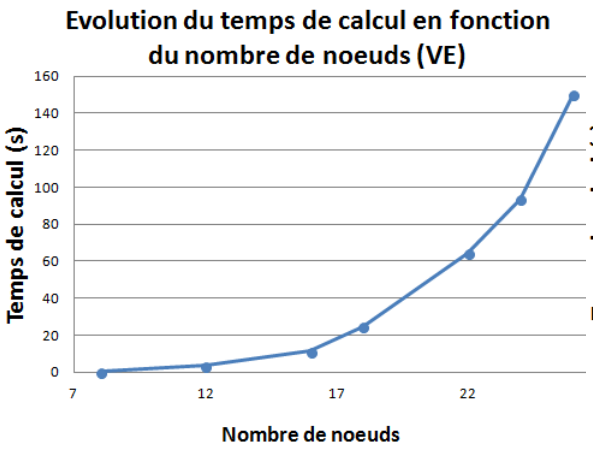


Figure 6.12 – Évolution du temps de calcul en fonction du nombre de nœuds (VE)

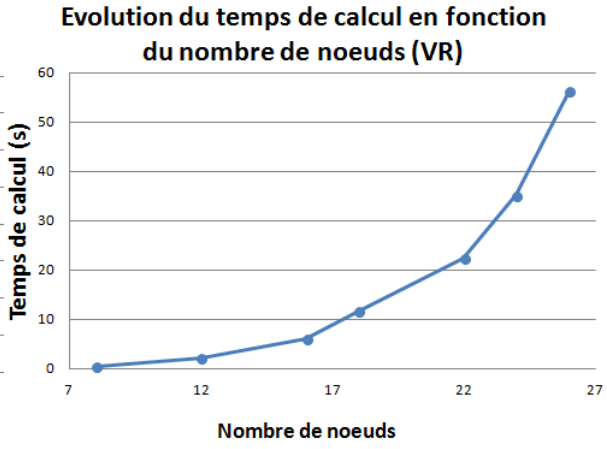


Figure 6.13 – Évolution du temps de calcul en fonction du nombre de nœuds (VR)

Le calcul du voisinage, c'est-à-dire le temps de calcul des coûts correspondant à l'application de tous les mouvements possibles est donc clairement dépendant de la taille de la structure en trois dimensions.

Les différences de temps de calcul entre un voisinage étendu et un voisinage restreint proviennent du fait que dans le voisinage étendu, nous effectuons une seule boucle contenant tous les mouvements possibles pour tous les arbres, alors que dans le cas du voisinage restreint, chaque arbre est optimisé séparément. Ce n'est donc pas tout à fait le même algorithme, d'où les différences de temps de calcul. Cependant, remarquons que les graphes ont la même forme, quel que soit le voisinage. En effet, l'augmentation du temps de calcul suit une fonction parabolique pour le voisinage étendu comme pour le voisinage restreint.

La sélection des paramètres pour l'algorithme de recherche avec tabous permet donc d'adapter l'algorithme à la structure des problèmes que nous avons choisi de résoudre. Avec une implémentation astucieuse qui permet de calculer rapidement le voisinage, nous obtenons des résultats en un temps raisonnable, compte tenu de la taille des problèmes.

CHAPITRE 7

CONCLUSION

L'affectation des composantes dans une architecture multiprocesseurs est un enjeu majeur dans le monde des nouvelles technologies. Dans ce mémoire, nous avons modélisé cette assignation par un problème d'affectation quadratique, ce qui nous a permis d'affecter les groupes de processeurs et les familles de mémoires aux nœuds de l'architecture. En biaisant la matrice de distance entre les nœuds , nous avons anticipé le dégagement de chaleur à l'intérieur de la structure pour essayer d'optimiser la répartition des flots dès la phase d'affectation des composantes.

Par la suite, nous avons introduit le problème de la sélection des bus que nous résolvons à l'aide d'une méthode de recherche avec tabous en définissant notamment deux voisinages, une liste de tabous de taille variable et deux critères d'aspiration dont le dernier permet une diversification de la recherche. Les résultats obtenus montrent qu'un voisinage étendu est plus adapté à la structure des problèmes que nous voulons résoudre, et qu'il est préférable de partir d'une bonne solution initiale pour obtenir les meilleurs résultats, même si la recherche avec tabous doit parfois s'éloigner d'une bonne solution pour explorer tout l'espace des solutions réalisables. Nous pourrions aussi exploiter les deux voisinages lors d'une résolution, par exemple en utilisant le voisinage restreint au début pour améliorer rapidement la mauvaise solution initiale puis en utilisant le voisinage étendu pour terminer la résolution.

Il serait intéressant de tester notre implémentation sur des problèmes de très grande taille pour vérifier les résultats obtenus et la prédominance de la méthode utilisant un voisinage étendu par exemple. D'autre part, l'utilisation de bus tels que ceux que nous pouvons trouver dans l'industrie rendrait notre étude encore plus proche des besoins que peuvent avoir les grandes entreprises dans l'affectation de composantes, notamment dans le domaine électronique. Nous pouvons aussi envisager la résolution des différents problèmes en utilisant d'autres heuristiques. Enfin, la littérature ne présente pas à notre

connaissance de problèmes résolus de façon optimale, ce qui nous permettrait de procéder à des comparaisons et d'offrir une analyse plus précise des résultats que nous avons obtenus.

BIBLIOGRAPHIE

[1] Aghaghi, Y., et al. Redundant Address BUS Encoding for Low Power. Proceedings of the International Symposium on Low Power Electronics and Design, 182-187, 2001.

[2] Ahadri, M.Z. La résolution du problème de formation de cellules dans un contexte multicritère. Mémoire de maîtrise, 2013.

[3] ALTERA Inc. Nios II Processor Reference Handbook. Version 10.1, www.altera.com, 2010.

[4] Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J. The Traveling Salesman Problem : A Computational Study. Princeton University Press, Princeton, 2007.

[5] ARC. Customizing a Soft Microprocessor Core. www.arc.com, 2002.

[6] Artieri, A., et al. NomadikTM Open Multimedia Platform for Nextgeneration Mobile Devices. STMicroelectronics Technical Article TA305, 2003.

[7] Asanovc, K., et al. The landscape of parallel computing research : a view from Berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, 2006.

[8] Banakar, R., et al. Scratchpad Memory : Design Alternative for Cache on-Chip Memory in Embedded Systems. Proceedings of CODES, 2002.

[9] Bononi, L., et al. NoC topologies exploration based on mapping and simulation models. Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 543-546, 2007.

[10] Burger, D., et al. Scaling to the end of silicon with edge architectures. Computer 37 (7), 44-55, 2004.

[11] Chen, T.-C., et al. NTUplace : a ratio partitioning based placement algorithm for large-scale mixedsize designs. Proceedings of ISPD, 236-238, 2005.

[12] Chou, C., Marculescu, R. Contention-aware application mapping for networkon-chip communication architectures. Proceedings of the International Conference on Computer Design, 2008.

[13] Dally, W.J., Towles, B. Principles and Practices of Interconnection Networks. Morgan Kaufmann, 2003.

[14] Del Cuvillo, J., et al., Towards a Software Infrastructure for Cyclops-64 Cellular Architecture. HPCS, 2006.

[15] Dobkin, R., et al., Fast Asynchronous Shift Register for Bit-Serial Communication. Proceedings of ASYNC, 2006.

[16] Dobkin, R., et al., High-Speed Serial Interconnect for NoC. NoC Workshop, DATE, 2006.

[17] Dorigo, M., Maniezzo, V., Colorni, A. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 1991.

[18] Dutta, S., et al., Viper : A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems. IEEE Design and Test 18 (5), 21-31, 2001.

[19] Dyer, M., et al. Design and Analysis of System on a Chip Encoder for JPEG2000. Circuits and Systems for Video Technology, IEEE Transactions 19 (2), 215-225, 2009.

[20] Élie, E. Approche efficace pour la conception des architectures multiprocesseurs sur puce électronique. Thèse de doctorat, 2010.

[21] Elshafei, A.N. Hospital layout as a quadratic assignment problem. Operations Research Quarterly 28 (1), 167-179, 1977.

[22] Fernandez, F.M. Introduction to Perturbation Theory in Quantum Mechanics. CRC Press first edition, 2000.

[23] Finke, G., Burkard, R.E., Rendl, F. Quadratic assignment problems. Annals of Discrete Mathematics 31, 61-82, 1987.

[24] Flachs, B., et al. The microarchitecture of the streaming processor for a Cell processor. IEEE Journal of Solid-State Circuits 41 (1), 63-70, 2006.

[25] Floyd, R.W. Algorithm 97 : Shortest path. Communications of the ACM, 5 (6), 345, 1962.

[26] Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research 13, 156-166, 1986.

[27] Ha, S., et al. PeaCE : a hardware-software codesign environment for multimedia embedded systems. ACM Transactions on Design Automaton of Electronic Systems, 1-25, 2007.

[28] Held, J., et al. From a few cores to many : A tera-scale computing research overview. Intel White Paper, 2006.

[29] Hennessy, J., Patterson, D.A. Computer Architecture : a Quantitative Approach. Morgan Kaufmann (3), 2003.

[30] Hu, J., Marculescu, R. Energy-aware mapping for tilebased NoC architectures under performance constraints. Proceedings of the 2003 conference on Asia South Pacific design automation, 233-239, 2003.

[31] Huerta, P., et al. A MicroBlaze Based Multiprocessor SoC. WSEAS Transactions on Circuits and Systems 4 (5), 423-430, 2005.

[32] Ienne, P., Leupers, R. Customizable Embedded Processors. MorganKauffman, 2006.

[33] Jensen, M.T. Reducing the run-time complexity of multiobjective EAs : the NSGA-II and other algorithms. IEEE Transactions on Evolutionary Computation 7 (5), 503-515, 2003.

[34] Jose, A.P., et al. Pulsed Current-Mode Signaling for Nearly Speed-of-Light Intrachip Communication. Journal of Solid-State Circuits 41 (4), 772-780, 2006.

[35] Kangmin, L., et al. SILENT : serialized low energy transmission coding for onchip interconnection networks. International Conference on Computer Aided Design, 448-451, 2004.

[36] Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P. Optimization by simulated annealing. Science 220, 671-680, 1983.

[37] Koopmans, T., Beckmann, M. Assignment problems and the location of economic activities. Econometrica 25, 53-76, 1957.

[38] Kwok, Y.-K., Ahmad, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computer Survey 31 (4), 406-471, 1999.

[39] Lahiri, K., et al. Design space exploration for optimizing onchip communication architectures. IEEE Transactions on Computer-Aided Design on Integrated Circuits and Systems 23 (6), 2004.

[40] Lambrechts, A., et al. Power Breakdown Analysis for a Heterogeneous NoC Platform Running a Video Application. IEEE International Conference 16, 179-184, 2005.

[41] Mak, R.H. Design and performance analysis of buffers : a constructive approach. Proceedings of ASYNC, 137-148, 2002.

[42] Metropolis, W., Roenbluth, A. and M., Teller, A. and E. Equation of the state calculations by fast computing machines. Journal of Chemical Physics 21, 1087-1092, 1953.

[43] Moore, G. E., Cramming more components onto integrated circuits. Electronics, 114-117, 1965.

[44] Ogoubi, E., et al. The Isometric on Chip Multiprocessor Architecture. Rapport Technique No. 1287, Université de Montréal, 2006.

[45] Parada, V., Ferland, J.A., Arias, M., Daniels, K. Optimization of Electrical Distribution Feeders Using Simulated Annealing. IEEE Transactions on Power Delivery 19, 1135-1141, 2004.

[46] Parada, V., Ferland, J.A., Arias, M., Schwarzenberg, P., Vargas, L. Heuristic Determination of Distribution Trees. IEEE Transactions on Power Delivery 25 (2), 861-869, 2010.

[47] Rogger, A., Fiechter, C.N., Werra, D. Basic ideas of tabu search with an application to traveling salesman and quadratic assignment. Ricerca Operativa 62, 5-28, 1992.

[48] Roy, J., et al. Satisfying whitespace requirements in top-down placement. Proceedings of ISPD, 206-208, 2006.

[49] Saastamoinen, I., et al. Interconnect IP for Gigascale System-on-chip. Proceedings of the European Conference Circuit Theory and Design, 281-284, 2001.

[50] Salminen, E., et al. HIBI-based multiprocessor SoC on FPGA. IEEE International Symposium 4, 3351-3354, 2005.

[51] Spindler, P., Johannes, F. M. Fast and robust quadratic placement combined with an exact linear net model. Proceedings of ICCAD, 2006.

[52] Stan, M. R., et al. Law-Power Encoding for Global Communication in CMOS VLSI. Transactions an VLSI Systems 5 (4), 1997.

[53] Susan, X., Pollitt-Smith, H. A Multi-MicroBlaze Based SOC System : From SystemC Modeling to FPGA Prototyping. IEEE/IFIP International Symposium on Rapid System Prototyping 19, 121-127, 2008.

[54] Suutari, T., et al. High-speed Serial Communication with Error Correction Using 0.25 ţm CMOS Technology. Proceedings of ISCAS 4, 618-621, 2001.

[55] Taghavi, T., et al. Dragon2006 : Blockage-aware congestion-controlling mixed-size placer. Proceedings of ISPD, 209-211, 2006.

[56] Taillard, E.D. FANT : Fast ant system. Technical report IDSIA-46-98, IDSIA, Lugano, 1998.

[57] Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J.-Y. La programmation à mémoire adaptative ou l'évolution des algorithmes évolutifs. Calculateurs Parallèles, 10 (2), 1998.

[58] Taillard, E.D. Robust taboo search for the quadratic assignment problem. Parallel Computing 17, 443-455, 1991.

[59] Tamhankar, R., et al. Performance driven reliable link design for networks on chips. Proceedings of the 2005 Asia and South Pacific Design Automation Conference, 749-754, 2005.

[60] Teifel, J., Manohar, R. A High-Speed Clockless Serial Link Transceiver. Proceedings of ASYNC, 151-161, 2003.

[61] Viswanathan, N., Chu, C.C.N. Fastplace : Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. IEEE Transactions on Computer-Aided Desig of Circuits and Systems 24 (5), 722-733, 2005.

[62] Walrand, J., Varaiya, P. High-Performance Communication Networks. Morgan Kaufman, 2000.

[63] Wentzlaff, D., et al. On-chip interconnection architecture of the tile processor. Micro, IEEE 27 (5), 15-31, 2007.

[64] Wilson D.B. Generating random spanning trees more quickly than the cover time. Proceeding STOC '96, 296-303, 1996.

[65] XILINX CORP. Xilinx MicroBlaze and PowerPC Processor Embedded Kit Virtex-5 FX70T FPGA Edition, Getting Started Guide. UG699 (v1.0.1), 2009.

[66] XILINX CORP. MicroBlaze Processor Reference Guide : Embedded Development Kit EDK 13.1. UG081 (v12.0), 2008.

[67] Xu, W., et al. A Quantitative Study of the On-Chip Network and Memory Hierarchy Design for Many-Core Processor. Parallel and Distributed Systems. IEEE International Conference 14, 689-696, 2008.

[68] Zhang, W.-T., et al. Design of heterogeneous MPSoC on FPGA. ASIC, ASICON. 7th International Conference, 102-105, 2007.

Annexe I

Matrice de flot F avec 18 nœuds utilisée avec le graphe de la section 4.4.2

0	0	0	0	0	0	0	0	0	713	10	170	0	0	7	233	10	147
0	0	0	0	0	0	0	0	0	0	627	8	251	5	35	6	0	0
0	0	0	0	0	0	0	0	0	18	13	440	6	98	18	16	22	7
0	0	0	0	0	0	0	0	0	22	409	7	875	0	93	28	111	15
0	0	0	0	0	0	0	0	0	0	0	275	9	1025	11	10	20	98
0	0	0	0	0	0	0	0	0	0	0	0	2	0	440	0	0	0
0	0	0	0	0	0	0	0	0	121	18	0	9	87	5	867	0	78
0	0	0	0	0	0	0	0	0	153	6	1	267	0	91	0	1246	0
0	0	0	0	0	0	0	0	0	376	8	0	0	50	52	84	26	1064
713	0	18	22	0	0	121	153	376	0	0	0	0	0	0	0	0	0
10	627	13	409	0	0	18	6	8	0	0	0	0	0	0	0	0	0
170	8	440	7	275	0	0	1	0	0	0	0	0	0	0	0	0	0
0	251	6	875	9	2	9	267	0	0	0	0	0	0	0	0	0	0
0	5	98	0	1025	0	87	0	50	0	0	0	0	0	0	0	0	0
7	35	18	93	11	440	5	91	52	0	0	0	0	0	0	0	0	0
233	6	16	28	10	0	867	0	84	0	0	0	0	0	0	0	0	0
10	0	22	111	20	0	0	1246	26	0	0	0	0	0	0	0	0	0
147	0	7	15	98	0	78	0	1064	0	0	0	0	0	0	0	0	0

Annexe II

Matrice d'incidence *A* avec 18 nœuds utilisée avec le graphe de la section 4.4.2

0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0

Annexe III

Matrice de distance *D* avec 18 nœuds obtenue après application de l'algorithme de Floyd-Warshall sur la matrice d'incidence de l'annexe II

0	1	2	1	2	3	1	2	3	2	3	4	2	3	4	3	4	5
1	0	1	2	1	2	2	1	2	3	2	3	3	2	3	4	3	4
2	1	0	3	2	1	3	2	1	4	3	2	4	3	2	5	4	3
1	2	3	0	1	2	2	3	4	1	2	3	3	4	5	2	3	4
2	1	2	1	0	1	3	2	3	2	1	2	4	3	4	3	2	3
3	2	1	2	1	0	4	3	2	3	2	1	5	4	3	4	3	2
1	2	3	2	3	4	0	1	2	1	2	3	1	2	3	2	3	4
2	1	2	3	2	3	1	0	1	2	1	2	2	1	2	3	2	3
3	2	1	4	3	2	2	1	0	3	2	1	3	2	1	4	3	2
2	3	4	1	2	3	1	2	3	0	1	2	2	3	4	1	2	3
3	2	3	2	1	2	2	1	2	1	0	1	3	2	3	2	1	2
4	3	2	3	2	1	3	2	1	2	1	0	4	3	2	3	2	1
2	3	4	3	4	5	1	2	3	2	3	4	0	1	2	1	2	3
3	2	3	4	3	4	2	1	2	3	2	3	1	0	1	2	1	2
4	3	2	5	4	3	3	2	1	4	3	2	2	1	0	3	2	1
3	4	5	2	3	4	2	3	4	1	2	3	1	2	3	0	1	2
4	3	4	3	2	3	3	2	3	2	1	2	2	1	2	1	0	1
5	4	3	4	3	2	4	3	2	3	2	1	3	2	1	2	1	0

Annexe IV

Matrice de distance $D_{modifiee}$ avec 18 nœuds utilisée avec le graphe de la section 4.4.2

0	20	40	1	21	41	20	89	60	21	90	61	40	60	80	41	61	81
20	0	20	21	41	21	40	69	40	41	110	41	60	80	60	61	81	61
40	20	0	41	21	1	60	89	20	61	90	21	80	60	40	81	61	41
1	21	41	0	20	40	21	90	61	20	89	60	41	61	81	40	60	80
21	41	21	20	0	20	41	110	41	40	69	40	61	81	61	60	80	60
41	21	1	40	20	0	61	90	21	60	89	20	81	61	41	80	60	40
20	40	60	21	41	61	0	69	80	41	110	81	20	40	60	21	41	61
89	69	89	90	110	90	69	0	69	110	100	110	89	69	89	90	110	90
60	40	20	61	41	21	80	69	0	81	110	41	60	40	20	61	41	21
21	41	61	20	40	60	41	110	81	0	69	80	21	41	61	20	40	60
90	110	90	89	69	89	110	100	110	69	0	69	90	110	90	89	69	89
61	41	21	60	40	20	81	110	41	80	69	0	61	41	21	60	40	20
40	60	80	41	61	81	20	89	60	21	90	61	0	20	40	1	21	41
60	80	60	61	81	61	40	69	40	41	110	41	20	0	20	21	41	21
80	60	40	81	61	41	60	89	20	61	90	21	40	20	0	41	21	1
41	61	81	40	60	80	21	90	61	20	89	60	1	21	41	0	20	40
61	81	61	60	80	60	41	110	41	40	69	40	21	41	21	20	0	20
81	61	41	80	60	40	61	90	21	60	89	20	41	21	1	40	20	0

Annexe V

Matrices d'interaction *IA* utilisées pour les résultats numériques

PR1 : 8 nœuds

1200	63	6	0
92	502	110	84
61	31	500	99
60	130	0	1067

PR2 : 12 nœuds

1843	25	20	18	35	216
8	1509	50	37	42	36
23	113	1838	36	29	47
37	374	257	1763	22	0
214	0	0	28	533	2
6	0	1	153	91	1513

PR3 : 16 nœuds

1129	139	0	0	116	0	0	108
38	285	83	0	70	0	13	51
126	0	804	141	41	0	0	0
110	0	55	905	0	120	141	41
120	94	78	16	1094	65	53	91
126	0	61	17	0	292	46	134
77	77	144	34	0	128	601	47
0	2	0	0	150	76	13	315

PR4 : 18 nœuds

713	10	170	0	0	7	233	10	147
0	627	8	251	5	35	6	0	0
18	13	440	6	98	18	16	22	7
22	409	7	875	0	93	28	111	15
0	0	275	9	1025	11	10	20	98
0	0	0	2	0	440	0	0	0
121	18	0	9	87	5	867	0	78
153	6	1	267	0	91	0	1246	0
376	8	0	0	50	52	84	26	1064

PR5 : 22 nœuds

880	101	77	0	0	9	0	73	137	115	96
0	320	0	0	141	0	30	17	2	0	0
0	0	526	27	131	98	50	95	0	0	108
135	0	62	234	65	41	118	5	124	11	124
65	0	108	0	642	0	0	0	128	92	0
147	76	29	144	110	211	43	95	84	73	144
99	87	0	121	91	46	866	61	21	105	16
29	122	0	22	0	95	0	296	127	28	48
117	110	0	108	101	0	62	29	960	0	127
0	137	147	119	128	101	46	71	5	406	0
0	0	100	103	123	68	0	46	142	38	457

PR6 : 24 nœuds

428	96	39	0	28	95	0	126	20	9	87	75
142	576	0	73	109	72	11	50	0	0	2	78
0	52	812	0	108	133	78	0	118	104	57	0
131	0	139	1146	4	119	112	0	36	0	30	0
71	0	72	0	297	0	12	0	0	29	138	136
74	50	43	144	49	774	138	4	30	109	33	69
61	53	84	0	149	138	529	0	0	43	0	120
142	89	150	120	149	31	124	561	88	0	44	0
108	126	43	0	83	0	46	65	594	18	120	58
0	0	96	0	125	68	143	99	126	289	11	0
142	0	0	0	53	147	80	97	0	62	865	19
0	38	0	58	92	44	0	128	27	90	65	559

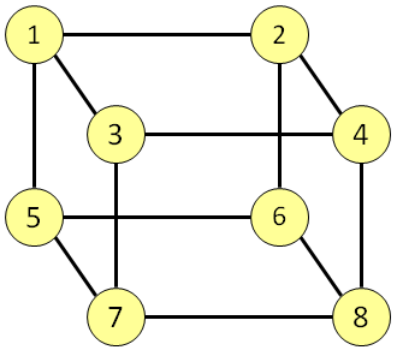
PR7 : 26 nœuds

720	71	6	141	83	63	52	3	74	35	63	59	0
0	516	57	134	3	0	76	25	147	0	0	52	52
0	0	335	34	141	0	124	35	120	0	63	67	108
76	0	0	1193	7	0	0	40	59	23	93	0	66
0	14	0	73	356	60	28	131	0	66	0	117	0
77	29	0	84	0	1026	107	57	122	62	0	48	150
52	119	0	107	0	62	443	82	132	0	0	48	101
20	89	23	87	100	101	148	859	99	30	0	55	36
145	0	0	0	83	143	93	0	242	37	11	0	100
0	142	27	37	0	81	135	143	88	756	0	136	0
78	66	98	21	0	122	91	144	55	58	330	0	85
0	139	73	44	142	0	7	131	104	135	97	1022	0
99	0	0	38	52	122	0	0	140	95	0	119	394

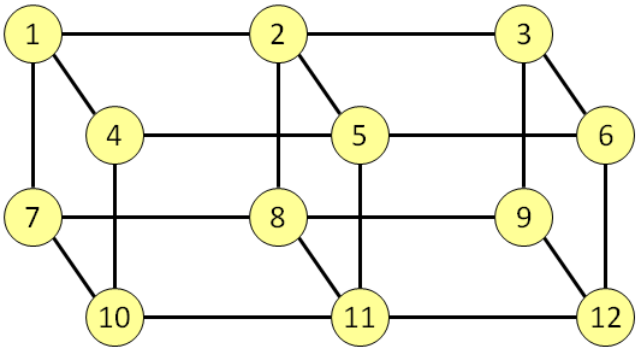
Annexe VI

Structure des graphes utilisés pour la résolution des problèmes

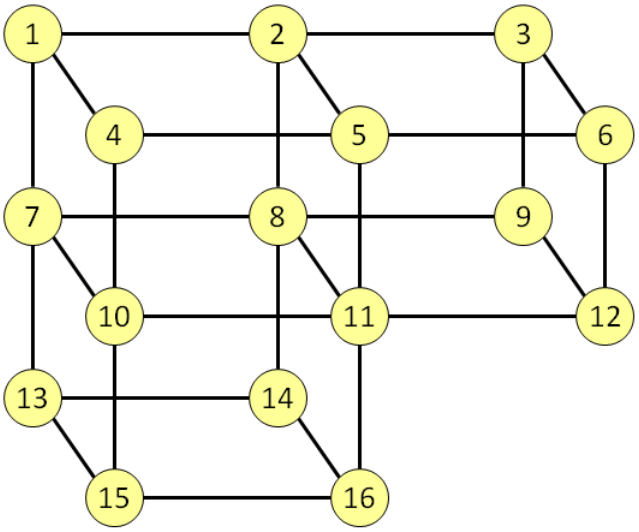
PR1



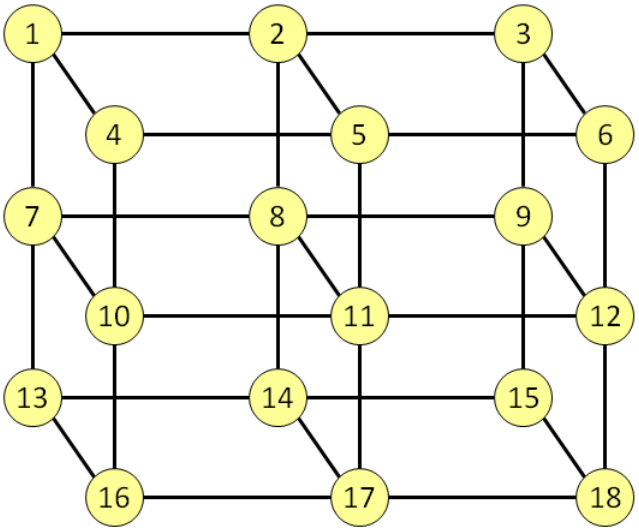
PR2



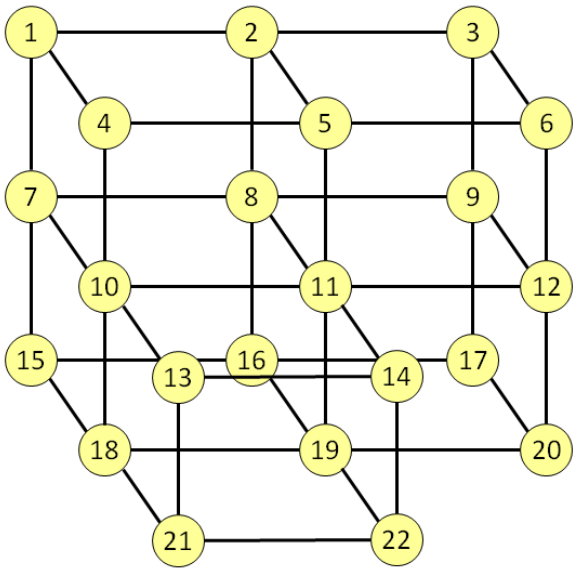
PR3


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

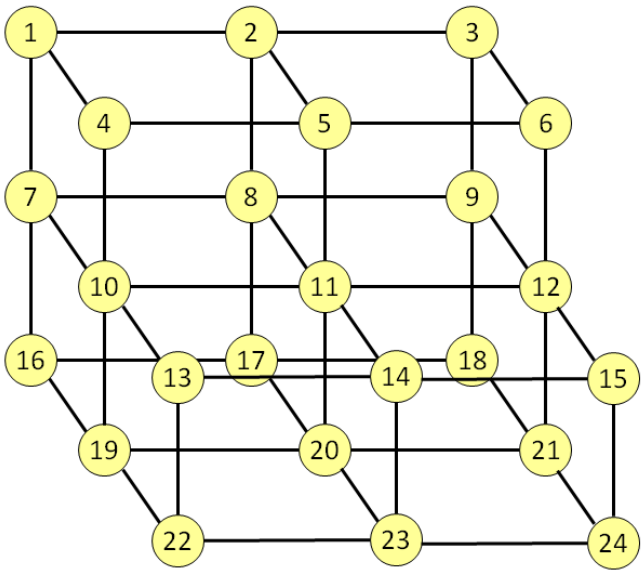
PR4


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

PR5


1
2
3
4
5
6
7
8
9
10
11
12
15
16
13
14
17
18
19
20
21
22

PR6


1
2
3
4
5
6
7
8
9
10
11
12
16
17
18
13
14
15
19
20
21
22
23
24

PR7