**République Algérienne Démocratique et Populaire**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**
**Université 8 Mai 1945 Guelma**

**Faculté** des **mathématiques et de l'informatique et des sciences de la matière**
**Département d'informatique**
**Laboratoire des sciences et technologies de l'information et de la communication**

# THÈSE

## EN VUE DE L'OBTENTION DU DIPLOME DE DOCTORAT EN 3ème CYCLE

Domaine : Mathématiques et informatique Filière : Informatique
Spécialité : Informatique

**Présenté par**

**Gattal Elhachemi**

***Intitulée***

**Modélisation et résolution du problème d'allocation de mémoire dans les systèmes embarqués comme problème d'affectation généralisée avec contraintes compliquantes**

Soutenue le : 06/06/2022 Devant le Jury composé de :

| Nom et Prénom | Grade | | |
|---|---|---|---|
| **Pr. NEMISSI Mohamed** | **Prof** | Univ. de 8 Mai 1945, Guelma | Président |
| **Pr. HADDADI Salim** | **Prof** | Univ. de 8 Mai 1945, Guelma | Directeur de thèse |
| **Pr. FARAH Nadir** | **Prof** | Univ. de Badji Mokhtar, Annaba | Examinateur |
| **Pr. BOUHADADA Tahar** | **Prof** | Univ. de Badji Mokhtar, Annaba | Examinateur |
| **Pr. KOUAHLA Med Nadjib** | **Prof** | Univ. de 8 Mai 1945, Guelma | Examinateur |
| **Dr. FERRAG Med Amine** | **MCA** | Univ. de 8 Mai 1945, Guelma | Examinateur |

**Année Universitaire** : 2021/2022

# Remerciements

Avant tout, je voudrais remercier Dieu de m'avoir donné la force, la connaissance, la capacité et l'opportunité d'entreprendre cette étude de recherche, de persévérer et de la terminer de manière satisfaisante.

Je tiens à remercier sincèrement mon directeur de thèse, Professeur **HADDADI Salim**, pour l'orientation remarquable de mes études supérieures. Je vous remercie de m'avoir enseigné et conseillé pendant ces quatre années. Vos conseils essentiels m'ont permis de tracer les grandes lignes de mes années suivantes en tant qu'étudiant de troisième cycle, en construisant ma future carrière de chercheur. Merci de m'avoir fait confiance et pour tout ce que vous avez fait pour m'amener jusqu'ici.

Je remercie beaucoup le directeur de laboratoire LabSTIC, le Professeur **SERIDI Hamid** qui m'a beaucoup aidé, encouragé et guidé depuis plusieurs années. Je le remercie pour ses nombreuses remarques et appréciations d'une extrême pertinence, et je tiens à lui exprimer ma profonde gratitude à cette occasion.

Je tiens à exprimer ma profonde gratitude aux Professeurs **NEMISSI Mohamed** et **KOUAHLA Med Nadjib** de l'Université de Guelma, aux Professeurs **BOUHADADA Tahar** et **FARAH Nadir** de l'Université de Badji Mokhtar, Annaba et le Docteur **FERRAG Med Amine** de l'Université de Guelma, pour l'honneur qu'ils m'ont fait en acceptant la responsabilité d'examiner ce travail et de participer au jury de soutenance.

Je tiens également à remercier tous les enseignants du département d'informatique de l'Université 8 mai 1945 Guelma pour la qualité de leur formation en particulier les Docteurs **KOUAHLA Zineddine**, **FAROU Brahim**, **HALLACI Samir**, et mon collègue le Docteur **BENRAZEK ALA-Eddine**. Un grand merci aux équipes du laboratoire LabSTIC qui ont contribué, chacune à leur manière, et je n'oublie pas l'ingénieur de laboratoire Mlle **Madiha KHAROUBI**.

# Résumé

La gestion de la mémoire dans les systèmes embarqués a un impact important sur les performances, en particulier, sur la consommation d'énergie. Comme l'allocation de mémoire (MAP) étant un processus difficile, elle est déléguée au compilateur. Toutefois, une optimisation minutieuse de l'allocation de la mémoire peut permettre de réaliser des économies importantes en termes de temps d'exécution et de consommation d'énergie. Notre étude se limite au problème de l'allocation des structures de données à la mémoire de manière à minimiser la consommation d'énergie. Ce dernier est modélisé comme un problème d'optimisation combinatoire NP-difficile.

L'objectif principal de cette thèse est de proposer de nouvelle approche pour la résolution du problème étudié capables de générer des résultats de bonne qualité, et de garantir que les méthodes suggérées sont génériques et applicables à d'autres problèmes d'optimisation.

Il comporte trois contributions :

— La première contribution présente la transformation du problème MAP en GAP, la conception d'une méthode de résolution rapide et suffisamment précise pour ce dernier est donc notre principale motivation.

— La deuxième contribution propose une méthode en deux étapes pour résoudre le GAP. La première étape est une heuristique de réduction des données pour réduire la taille du problème. La deuxième étape utilise un solveur MIP (Cplex) pour résoudre le GAP réduit (RGAP).

— La troisième contribution présente une méthode basée sur l'algorithme de recherche locale itérée ; l'ILS est appliqué aux données introduites par l'heuristique de réduction des données (RGAP) présentée ci-dessus, puis les solutions élites générées par l'ILS sont utilisées pour réduire la taille du problème qui sera ensuite résolu facilement et très rapidement par un solveur MIP (Cplex).

L'approche proposée a été testée sur un ensemble de données d'instance standard à grande échelle disponible sur le site web : www.al.cm.is.nagoya-u.ac.jp/~yagiura/

gap. Les résultats obtenus sont très intéressants et la comparaison montre que l'approche proposée est compétitive par rapport à l'état de l'art.

**Mots-clés** : Système embarqué, Allocation mémoire, Problème d'affectation généralisé, méta-heuristique, Réduction de donnée, Recherche locale itérée, Solveur MIP.

# Abstract

Memory management in embedded systems has a significant impact on performance, in particular, on power consumption. Because memory allocation (MAP) is a difficult process, it is delegated to the compiler. However, careful optimization of memory allocation can result in significant savings in runtime and power consumption. Our study is limited to the problem of allocating data structures in memory to minimize energy consumption. The latter is modeled as an NP-hard combinatorial optimization problem.

The main objective of this thesis is to propose a new approach for the resolution of the studied problem capable of generating good quality results and guarantee that the suggested methods are generic and applicable to other optimization problems.

It has three contributions :

— The first contribution presents the transformation of the MAP problem into a GAP, the design of a fast and sufficiently precise resolution method for the latter is our main motivation.

— The second contribution proposes a two-step method to solve the GAP. The first step is a data reduction heuristic to reduce the size of the problem. The second step uses a MIP solver (Cplex) to solve the reduced GAP (RGAP).

— The third contribution presents a method based on the iterated local search algorithm ; the ILS is applied to the data introduced by the data reduction heuristic (RGAP) presented above, and then the elite solutions generated by the ILS are used to reduce the size of the problem which will then be solved easily and very quickly by a MIP solver (Cplex).

The proposed approach was tested on a large-scale standard instance dataset available on the website : www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap. The results obtained are very interesting, and the comparison shows that the proposed approach is competitive with respect to the state of the art.

**Keywords** : Embedded system, Memory allocation, Generalized allocation problem, Meta-heuristic, Data reduction, Iterated local search, MIP solver.

# ملخص

إدارة الذاكرة في الأنظمة المضمنة لها تأثير كبير على الأداء، ولا سيما على استهلاك الطاقة. نظرًا لأن تخصيص الذاكرة (MAP) عملية صعبة، يتم تفويضها إلى المترجم. ومع ذلك، يمكن أن يؤدي التحسين الدقيق لتخصيص الذاكرة إلى توفير كبير في وقت التشغيل واستهلاك الطاقة. تقتصر دراستنا على مشكلة تخصيص هياكل البيانات في الذاكرة لتقليل استهلاك الطاقة. تم تصميم هذا الأخير على أنه مشكلة من مشاكل التحسين التوافقي الصعبة من الناحيتين النظرية والعلمية.

الهدف الرئيسي من هذه الأطروحة هو اقتراح نهج جديد لحل المشكلة المدروسة قادر على تحقيق نتائج جيدة النوعية ، ولضمان أن تكون الطرق المقترحة عامة وقابلة للتطبيق على مشاكل التحسين الأخرى.

تشمل ثلاث مساهمات:

- تقدم المساهمة الأولى كيف يصبح نموذج تخصيص البيانات الى الذاكرة في الأنظمة المضمنة كنموذج مشكلة التخصيص المعممة. لذلك فإن تصميم طريقة حل سريعة ودقيقة بما فيه الكفاية لهذا الأخير هو الدافع الرئيسي لدينا.
- تقترح المساهمة الثانية طريقة من خطوتين لحل مشكلة التخصيص المعممة. الخطوة الأولى هي إرشاد تقليل البيانات لتقليل حجم هذه المشكلة. تستخدم الخطوة الثانية أداة حل تجارية تسمى السبلاكس لحل مشكلة التخصيص المعممة التي تم تقليل بياناتها (جاب مصغر).
- تقدم المساهمة الثالثة طريقة تعتمد على خوارزمية البحث المحلي المتكرر؛ يتم تطبيق هذه الخوارزمية على البيانات المقدمة من خلال دليل تقليل البيانات (الجاب المصغر) المقدم أعلاه، ثم يتم استخدام حلول النخبة التي تم إنشاؤها بواسطة خوارزمية البحث المحلي المتكرر لتقليل حجم المشكلة حيث يتم حلها بعد ذلك بسهولة وبسرعة كبيرة بواسطة محلل السبلاكس.

تم تقييم النهج المقترح على مجموعة بيانات مثيل معيارية واسعة النطاق متاحة على موقع الويب: www.al.cm.is.nagoya-u.ac.jp/yagiura/gap . النتائج التي تم الحصول عليها مثيرة للاهتمام للغاية وتظهر المقارنة أن النهج المقترح منافس لأحدث ما توصلت إليه الأبحاث في هذا المجال.

**الكلمات المفتاحية:** مشكلة تخصيص الذاكرة، مشكلة التخصيص المعممة، الاستدلال الفوقي، تقليل البيانات، البحث المحلي المتكرر، تحسين التدرج الفرعي.

# Table des matières

# Table des figures

# Liste des tableaux

# Acronymes

| | |
|---|---|
| **OC** | **O**ptimisation **C**ombinatoire |
| **PO** | **P**roblème d'**O**ptimisation |
| **POC** | **P**roblème d'**O**ptimisation **C**ombinatoire |
| **RL** | **R**elaxation **L**agrangienne |
| **PR** | **P**roblème **R**elaxé |
| **DL** | **D**ual **L**agrangien |
| **PL** | **P**rogrammation **L**inéaire |
| **PB** | **P**rogramme **B**inaire |
| **TC** | **T**héorie de la **C**omplexité |
| **PLNE** | **P**rogramme **L**inéaire en **N**ombres **E**ntiers |
| **GAP** | **G**eneralized **A**assignment **P**roblem |
| **RGAP** | **R**eduction **GAP** |
| **MAP** | **M**emory **A**llocation **P**roblem |
| **LS** | **L**ocal **S**earch |
| **ILS** | **I**terated **L**ocal **S**earch |
| **VLSN** | **V**ery **L**arge **S**cale **N**eighborhood |
| **VFH** | **V**ariable **F**ixing **H**euristic |
| **PPA** | **P**lant **P**ropagation **A**lgorithm |

# Introduction Générale

La recherche opérationnelle (RO), également connue sous le nom aide à la décision, est la science des processus et des méthodes permettant de prendre de meilleures décisions. C'est un ensemble de méthodes et de techniques visant à résoudre des problèmes d'optimisation (programmes mathématiques qui minimisent ou maximisent un ou plusieurs critères, tout en respectant certaines conditions appelées "contraintes") qui modélisent des problèmes réels dans plusieurs domaines (économie, finance, gestion, transport, logistique, communication, etc.). En résumé, la RO est très liée à l'ingénierie des systèmes.

L'optimisation combinatoire (OC) est une discipline de l'optimisation en mathématiques appliquées et en informatique, et a également été associée à la recherche opérationnelle, aux algorithmes et à la théorie de la complexité.

Le problème de l'optimisation combinatoire (POC) consiste à trouver la meilleure solution parmi l'ensemble des solutions possibles (ensemble discret). En général, cet ensemble est limité, mais comprend un très grand nombre d'éléments et est décrit implicitement, c'est-à-dire avec une liste relativement courte de contraintes que les solutions possibles doivent respecter.

Le système de mémoire est la principale source de performance et de consommation d'énergie des systèmes embarqués. Selon une taxonomie qui a été proposée dans plusieurs littératures, il y a beaucoup de techniques d'optimisation qui ont été appliquées à la mémoire, comme suit : optimisation matérielle, optimisation logicielle et optimisation matérielle/logicielle, et grâce à ces techniques, la consommation d'énergie des systèmes embarqués a été réduite.

Le problème d'allocation de mémoire (MAP) dans les systèmes embarqués (affection des structures de données en mémoire) constitue un défi majeur pour les concepteurs de systèmes embarqués. L'allocation de mémoire prend généralement en compte de nombreux paramètres : consommation d'énergie, performance, surface utilisée, coûts, etc. Notre étude se restreint au problème d'allocation des structures de données à la mémoire de manière à minimiser la consommation d'énergie. Ce dernier se modélise comme un problème d'optimisation combinatoire. En microélectronique, ce problème complexe est mal abordé pare ce que les concepteurs ne maîtrisent pas les outils mathématiques de l'optimisation discrète. Ce n'est pas sans raison qu'en France le CNRS a créé une unité de recherche, appelée GDR-RO, dont les thèmes de recherche sont les ≪ Problématiques d'optimisation discrète en micro-électronique ≫. L'approche préférée, dans la mesure du

possible, est un algorithme exact obtenant une solution optimale. Une approche alternative serait de proposer une ≪ bonne ≫ méta-heuristique.

L'objectif global de l'allocation de mémoire est l'allocation des structures de données d'une application spécifique à une architecture mémoire donnée avec des contraintes sur les bancs de mémoire et les structures de données. De plus, des mémoires externes sont maintenant présentes dans l'architecture cible pour stocker des données à long terme, et elles améliorent les performances des systèmes embarqués. L'allocation de mémoire vise à rechercher les structures de données afin de réduire le temps d'accès à ces données.

Dans cette thèse, on considère le MAP comme un GAP. GAP, ses diverses restrictions telles que le sac à dos, le sac à dos multiple, le conditionnement de bacs et ses extensions telles que le goulot d'étranglement, le multi-niveau, le stochastique et le quadratique GAP, constituent des problèmes d'optimisation combinatoire fondamentaux qui ont attiré beaucoup d'attention. GAP a servi de banc d'essai pour la plupart des paradigmes de recherche opérationnelle.

La thèse est organisée comme suit.

Le chapitre 1, explique le contexte scientifique de la thèse. Nous commençons par une introduction à l'optimisation combinatoire et à certains problèmes classiques de celle-ci, puis nous présentons quelques concepts de base de la théorie de la complexité.

Le chapitre 2, présente un résumé de toutes les méthodes de résolution utilisées : la relaxation lagrangienne et la méthode du sous-gradient, les notions de voisinage et d'amélioration locale, la recherche locale itérée et d'autres métaheuristiques en général.

Dans le chapitre 3, nous donnons un aperçu des techniques d'optimisation de la mémoire pour les systèmes embarqués. Nous avons passé en revue quelques techniques d'optimisation de la mémoire, ces techniques étant axées soit sur l'optimisation matérielle, soit sur l'optimisation logicielle, l'optimisation des liaisons de données (optimisation matérielle/logicielle).

Toujours dans ce chapitre 4, nous avons exposé le problème de l'allocation de mémoire en présentant les solutions proposées dans la littérature qui s'intéresse à ce domaine, et nous proposons un état de l'art.

Le chapitre 5 présente le problème d'affectation généralisé (GAP), certaines de ses applications pratiques et de ses propriétés, motivant l'approche que nous proposons. La méthode proposée commence par une heuristique de réduction des données pour réduire la taille de GAP afin de transformer le problème et le rendre plus facile à résoudre rapidement par un solveur MIP (`Cplex`). Ensuite, nous appliquerons la recherche locale ILS en combinant l'heuristique de réduction des données pour obtenir un sortant qui sera amélioré par la recherche locale en utilisant la matrice de coûts originale. La méthode proposée est testée sur un benchmark largement utilisé d'instances lourdes et volumineuses disponible sur le site : www.al.cm.is.nagoya-u.ac.jp/ yagiura/gap. Les résultats obtenus justifient l'efficacité de la méthode proposée. La méthode a fait l'objet d'une publication dans la revue "*International Journal of Management Science and Engineering Management (TMSE)*".

Enfin, nous terminons par une conclusion qui explique notre contribution et suggère quelques pistes de recherches futures.

# Chapitre 1

# Présentation des problèmes d'optimisation combinatoire

## 1.1 Introduction

L'optimisation joue un rôle important et croissant dans de nombreux domaines, tels que la conception de systèmes mécaniques, la conception de circuits électroniques, le traitement d'images et la recherche opérationnelle... etc, car elle vise à répondre aux exigences croissantes du secteur industriel et économique (maximisation des performances, minimisation des coûts).

L'optimisation combinatoire (OC) est une discipline qui combine l'informatique et les mathématiques appliquées et en tire son contexte général. L'OC consiste à étudier et à trouver une ou plusieurs solutions à un problème. On cherche à définir la meilleure solution (optimale) d'un ensemble de variables contraintes ; par rapport à une norme spécifique (minimiser ou maximiser la fonction à optimiser). Par exemple, chercher à maximiser l'efficacité d'une unité de production industrielle, fixer un temps de fabrication minimum, etc. Ce sont toutes des problèmes d'optimisation.

En effet, le principe de l'optimisation combinatoire est de rechercher un groupe distinct dans un sous-ensemble des meilleurs sous-groupes pour maximiser ou minimiser une fonction (ou plusieurs fonctions) sous certaines contraintes, afin de trouver la meilleure combinaison au milieu d'un grand nombre de possibilités [42].

Résoudre un problème d'optimisation consiste à rechercher le couple (x, y) qui, selon le cadre maximise (ou minimise) la fonction à optimiser.

## 1.2 Définition

Dans les problèmes d'optimisation combinatoire, les domaines de valeurs des variables sont discrets. On retrouve également dans l'espace de recherche, les contraintes associées au problème et on intègre le concept de distance entre les solutions possibles.

Le problème d'optimisation combinatoire POC est un problème de recherche qui implique l'exploration de l'espace de recherche. Un espace de recherche est défini comme un ensemble discret de solutions possibles à un problème. La solution correcte sera proche du point optimal de cet espace, sinon aussi proche que possible de l'optimum, ce qui permet de minimiser ou de maximiser une fonction objectif.

Un POC est défini sous la forme suivante :

$$f : X \longrightarrow \mathbb{R}, \text{ à trouver :}$$

— son minimum $v$ (resp. son maximum) dans $X$

— un point $x \in X$ qui réalise ce minimum (resp. maximum) i.e. $f(x) = v$.

écriture du problème : $min_{x \in X} f(x)$ resp. $max_{x \in X} f(x)$.

Où $f$ est la fonction objectif à valeurs réelles, qui évalue les solutions symbolisées par $x$.

Il s'agit de minimiser (ou maximiser) la fonction $f$ sur l'ensemble de toutes les solutions réalisables note $X$.

$v$ est la valeur optimale.

Les solutions $x$ ($x \in X$) ont une nature mathématique très diverse, car elles sont considérées comme un vecteur d'un espace de recherche à $n$-dimensionnel et $f$ désignera une fonction de cet espace dans $\mathbb{R}$. Quant à définir l'ensemble des solutions possibles $X$, cela se fait généralement par des contraintes souvent exprimées sous forme d'inégalités [167].

**Remarque** : la recherche maximale peut toujours se ramener à la recherche minimale. Liaison minimum/maximum : Soit $f$ une fonction dont on veut trouver un maximum. Le problème $max_{x \in X} f(x)$ retourne $(x, v)$ tandis que $min_{x \in X} f(x)$ retourne $(x, -v)$ .

le POC probablement facile (résoluble en temps polynomial) ou dur, selon la nature de l'ensemble $X$ et de la fonction $f$. Par exemple, si $X$ est un polyèdre convexe $\subseteq \mathbb{R}^n$ et $f$ une fonction linéaire, on retrouve un PL résoluble efficacement par l'algorithme du simplexe ou même par d'autres algorithmes polynomiaux. Par contre, si $X \subseteq \mathbb{Z}^n$, on retrouve des Programmes Linéaire en Nombre Entier (PLNE) qui constituent des problèmes difficiles ; aucun algorithme polynomial n'a encore été trouvé pour les résoudre [40].

Lorsqu'on présente une solution à un problème d'optimisation, il est nécessaire de définir clairement à quelle catégorie appartient ce problème. Certains algorithmes sont spécialement conçus pour résoudre un certain type de problème et ces algorithmes ne sont pas efficaces pour un autre type.

La classification des problèmes d'optimisation est différente selon les auteurs. Par exemple, on distingue selon [46] :

— Les problèmes d'optimisation à variables continues : si les variables peuvent prendre n'importe quelle valeur (réelle). En général, ces problèmes sont faciles à résoudre.

— Les problèmes d'optimisation discrète : le problème optimisation est dit discret, si les variables de décision sont discrètes (entières ou de binaires).

On dit que le problème d'optimisation en variables mixtes, s'il combine des variables continues et discrètes.

— Les Problèmes d'optimisation multi-critère, multidimensionnel ou multi choix : une recherche entre plusieurs objectifs contradictoires. En revanche, les problèmes à objectif unique sont identifiés par une seule fonction objectif.

— Les Problèmes statistiques ou dynamiques : si les données sont connues parfaitement, a implique que "optimisation est statistique", si ce n'est pas le cas, on dit "optimisation dynamique".

Il existe une autre classification qui divise les problèmes d'optimisation en problèmes avec ou sans contraintes. Il convient de noter que la résolution des problèmes avec contraintes est la plus complexe et nécessite des algorithmes personnalisés pour les résoudre.

De ce fait, le PO est caractérisé par :

(•) L'espace de recherche permet de déterminer le type d'optimisation. L'espace de recherche est défini par l'ensemble des domaines de définition des variables du problème. Les variables du problème peuvent être logique, réelles, entières, etc., et sont exprimées sous forme de données qualitatives ou quantitatives. Ainsi qu'une ou plusieurs fonctions objectifs et un ensemble de contraintes.

(•) Une fonction objectif comme but de l'étude du problème d'optimisation est de minimiser ou maximiser cette fonction, pour déterminer la meilleure solution d'un domaine de solutions potentielles.

(•) Un espace de recherche est généralement limité par un ensemble de contraintes, ces contraintes dans l'espace de recherche déterminent les conditions que les variables doivent remplir.

(•) Une méthode d'optimisation recherche sur une valeur optimale ou optimum.

La principale difficulté des problèmes d'optimisation combinatoire vient de l'absence de définition de la meilleure solution, car on peut seulement prouver que cette solution est meilleure que les autres.

## 1.3 Complexité et classification

La théorie de la complexité couvre deux aspects :

(•) La complexité des algorithmes : évaluée par la taille des données, le nombre d'opérations de l'algorithme et le temps d'exécution.

(•) La complexité des problèmes et leur classification.

### 1.3.1 Complexité

La complexité algorithmique est une estimation du temps qu'il faudrait à un algorithme pour se terminer, sachant que l'entrée est de taille $n$. Si un algorithme se doit d'être évolutif, il doit donner le résultat dans une limite de temps déterminée et pratique, même pour des valeurs significatives de $n$. Pour cette dernière raison, la complexité est calculée de façon asymptotique lorsque $n$ s'approche de l'infini. Si la complexité est souvent évaluée en termes de temps, elle est parfois également analysée en termes d'espace, ce qui se traduit par les exigences de l'algorithme en matière de mémoire.

L'analyse de la complexité d'un algorithme est nécessaire pour évaluer les algorithmes ou chercher à les perfectionner. La complexité algorithmique fait partie d'une des disciplines de l'informatique théorique appelée théorie de la complexité computationnelle.

La rédaction de cette section est basée essentiellement sur la référence [61].

Selon les caractéristiques du groupe des solutions, les problèmes peuvent être classés comme suit [43, 205] :

— Problème de décision :

  En algorithmique, un problème de décision est une question mathématiquement très spécifique qui est traitée par un programme informatique (sans restriction de

mémoire ou de temps), ce qui en fait une question à résoudre sous une forme précise liée aux paramètres présentés et exige une réponse positive ou négative. Cependant, certains problèmes de décision spécifiques ne peuvent être résolus en pratique en raison de la complexité des calculs. La théorie de la complexité algorithmique donne une hiérarchie des complexités formelles. En particularité, le problème NP-complet n'aura pas de solution effective en pratique, sauf dans certains cas particuliers de taille suffisamment petite [74].

— Problème polynomial réductible :

Est une méthode permettant de résoudre un problème à l'aide d'un autre [116].

Soit $A_1$ et $A_2$ deux problèmes, $A_1$ peut être réduit à $A_2$ à condition qu'il y ait une solution de l'algorithme $A_1$ qui utilise la solution de l'algorithme $A_2$. Dans le cas où l'algorithme $A_1$ résout un polynôme, considérant les appels à la solution de l'algorithme $A_2$ comme de complexité constante, la sténographie s'appelle un polynôme. On dit que $A_1$ est polynomial réductible en $A_2$ et on le note $A_1 \propto A_2$ .

— Problème de la classe P ou PTIME ($P \subseteq NP$) :

C'est la catégorie des problèmes faciles. Un problème avec P, s'il existe un algorithme déterministe qui le résout en temps polynomial [74].

— Problème de la classe NP (non déterministe polynomial) :

Classe des problèmes qui peut être résolue en temps polynomial par un algorithme non déterministe [74].

— Problème de la classe NP-complets :

Nous disons qu'un problème est dans la classe NP-complet si tout autre problème dans NP est polynomialement réductible dans cette dernière [74].

— Problème de la classe NP-difficiles (NP-durs) :

Est un problème qui n'admet pas d'algorithmes capables de trouver une solution en temps polynomial. Nous avons quelques exemples de problèmes NP-complets comme le problème des 3-colorants, le problème des cycles hamiltoniens, le problème du sac à dos, etc. [219].

Lorsqu'on trouve un algorithme efficace pour résout un problème, ce dernier est facile, car la facilité de résolution d'un problème est liée à l'existence d'un algorithme efficace qui fournit une solution, et parmi les problèmes faciles on trouve la programmation linéaire, le plus court chemin, ainsi que l'affectation, etc. Lorsqu'il existe un problème qui n'a pas d'algorithme de solution ou si un algorithme

de solution efficace ne peut être trouvé pour ce problème, on dit que le problème est difficile, c'est-à-dire qu'il appartient à la catégorie des problèmes NP-complets, parmi les problèmes "difficiles".

Ainsi par exemple, pour expliquer la complexité de l'algorithme A, la complexité est une fonction $Comp(N)$ qui donne le nombre d'instructions que l'algorithme A exécute dans le pire des cas pour des données de taille $N$. En général, la complexité est dépendante du mauvais cas. Pour plus d'informations voir les références [165, 205].

En outre, la découverte des mauvais cas est très utile pour les applications en temps réel. La complexité peut, dans le pire des cas, obscurcir notre vision de l'efficacité de l'algorithme. Le simplexe, par exemple, malgré la plus mauvaise complexité qu'il contient, est considéré comme un très bon algorithme pour résoudre la plupart des cas de programmes linéaires [51].

On distingue en général des algorithmes polynomiaux (complexité de l'ordre d'un polynôme par exemple : $\log n$, $n^{0.5}$, $n \log n$, $n^2$), et des algorithmes exponentiels ($e^n$, $2^n$,$n \log n$, la fonction factorielle $n!$).

Un bon algorithme est polynomial. Ce critère d'efficacité est confirmé par la pratique :

($*$) Une exponentielle dépasse tout polynôme pour n assez grand.

($*$) De très grands algorithmes polynomiaux peuvent être construits à l'aide d'algorithmes relativement petits. C'est le résultat d'opérations mathématiques appliquées aux polynômes telles que la multiplication, la division et la combinaison. Elles produisent des polynômes qui ont de bonnes et intéressantes propriétés de fermeture.

**POC faciles (polynomiaux)** [166] :

Max-flow : l'algorithme de Ford-Fulkerson permet de résoudre ce problème en $O(NM^2)$.

Min-cost path, shortest path : dans le cas des problèmes de graphes, si le graphe G et les coûts C sont quelconques, on dispose alors de l'algorithme de Bellman en $O(NM)$. L'algorithme de Dijkstra, en $O(N^2)$, est applicable quand les coûts de C sont positifs.

Minimum spanning tree : l'algorithme de Prim en $O(N^2)$ et celui de Kruskal, en $O(M \log_2 N)$.

Planarity testing : algorithme compliqué conçu par Hopcroft et Tarjan en $O(N)$.

Linear programming : La programmation linéaire est un outil mathématique très puissant utilisé pour la modélisation, et pour résoudre la programmation linéaire, on utilise l'algorithme du simplexe donc c'est un algorithme exponentiel. L'algorithme du simplexe est excellent en moyenne, mais Minty [137] a construit des cas où l'algorithme visite presque toutes les solutions de base (de l'ordre de $C_n^m$).

### 1.3.2 Classification

La classification des différents problèmes d'optimisation se fait généralement en fonction de leurs caractéristiques [75] :

(•) Nombre de variables de décision : problème mono-variable (une variable), problème multi-variable (plusieurs variables).

(•) Type de variable de décision : réel continu, entier ou discret, combinatoire.

(•) Type de fonction objectif : une fonction linéaire des variables de décision (problème linéaire), une fonction quadratique des variables de décision (problème quadratique), une fonction non linéaire des variables de décision (problème non linéaire).

(•) Formulation de problème : avec contraintes (problème à contraintes), sans contraintes (problème non contraint).

Au cours des années 1970, la théorie de la complexité (TC) qui a été développée, constitue une partie de la théorie de l'informatique, pour trouver des algorithmes appropriés pour résoudre un type spécifique de problème. Elle traite des ressources nécessaires au processus de calcul. Les plus courantes de ces ressources sont le temps (c'est-à-dire le nombre d'étapes ou le temps nécessaire pour résoudre le problème) et l'emplacement (c'est-à-dire la quantité de mémoire nécessaire pour résoudre le problème). D'autres ressources peuvent être prises en compte, telles que : combien de processeurs parallèles sont nécessaires pour effectuer le calcul en utilisant la programmation parallèle.

La théorie de la complexité est principalement basée sur les outils de la logique mathématique, elle ne traite que des problèmes d'existence (PE) dont la réponse est oui-non. Le POC est au moins aussi difficile que le problème d'existence.

Dans leur étude Garey [75] ont distingué trois catégories principales de complexité, qui ont été expliquées précédemment. La classe NP comprend tous les problèmes dont

la vérification peut être déterminée par un algorithme non déterministe en temps polynomial par rapport à la taille de l'instance. Bien sûr, les problèmes NP sont tous les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant avec un algorithme polynomial. Il existe deux catégories dans cette classe, les problèmes NP-complets et les problèmes NP-difficiles. Ces derniers sont les problèmes les plus difficiles de NP. La transformation polynomiale d'un problème est à l'origine du concept de problème NP-complet. De nombreux problèmes intéressants sont NP-complets et il n'existe pas encore de solution efficace, en raison du déterminisme. Il convient de noter que tous les problèmes NP ont été traités à grande échelle.

L'hypothèse qu'un problème peut être résolu par un algorithme polynomial non-déterministe signifie que pour une solution donnée, il est facile de vérifier au cours du temps du polynôme si elle correspond à un problème pour une situation donnée ; Mais ce nombre de solutions à tester pour résoudre le problème est exponentiel par rapport à la taille de l'instance. L'indéterminisme permet de masquer la taille exponentielle des solutions à tester tout en permettant à l'algorithme de rester polynomial.

Les problèmes NP-complets appartiennent aux problèmes NP-difficile, et d'un autre côté, un problème NP-difficile n'est pas nécessairement dans NP.

Voici un diagramme d'Euler montrant les relations et la différence entre NP, NP-complet et NP-difficile (NP-hard)(1.1) :

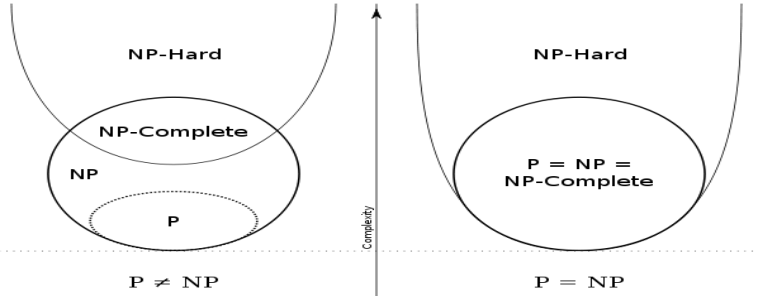


FIGURE 1.1 – Diagramme d'Euler

## 1.4 L'optimisation mono-objectif et optimisation multi-objectif

Il existe deux manières d'optimisations :

La première est le problème d'optimisation à mono-objectif, est généralement exprimé sous la forme d'un problème de minimisation ou de maximisation, et écrit sous la forme PLNE [163].

La solution optimale qui présente le coût optimal (minimum, maximum) peut être clairement définie. Officiellement, chaque cas de ce problème concerne un ensemble de $S$ de solutions potentielles qui respectent certaines limites et la fonction objectif $S \in f$, qui associe à chaque solution réalisable $s$ de $S$ une valeur $(s)$. La solution d'instance $(S)$ du problème d'optimisation consiste à trouver la solution optimale $s^*$ pour $S$ qui optimise la valeur de la fonction objectif du cas de minimisation : le but est de trouver $s^*$ de $S$ tel que $(s^* \in s)$ pour tout élément $s$ de $S$. Le problème maximum peut être défini de la similaire manière.

La deuxième manière est l'optimisation multi-objectifs simultanée. En optimisant un grand nombre de fonctions objectifs souvent antithétiques, nous essayons de trouver la meilleure solution en fonction d'un ensemble de performances du problème (temps de réponse plus fort et temps de réponse plus temps de collecte. Durabilité, etc). Le résultat d'un problème d'optimisation multi-objectifs est généralement une variété de solutions caractérisées par différents compromis entre les objectifs. Cette variété est connue sous le nom de Pareto-Optimal. L'objectif de l'optimisation multi-objectifs est donc d'obtenir des solutions de Pareto et de connaître l'ensemble des compromis possibles entre les objectifs.

**Problème d'optimisation multi-objectifs :**

Le problème de l'optimisation multi-objectifs a été étudié dans de nombreuses littératures : ([77, 210, 230]).

Les problèmes réels nécessitent plusieurs mesures ou objectifs de performance qui doivent être optimisés simultanément. Mais, cela n'est pas toujours possible en pratique, car les objectifs peuvent être contradictoires, puisqu'ils mesurent différents aspects de la qualité d'une solution. Dans ce cas, la qualité de l'individu n'est pas décrite par une échelle, mais par un vecteur : Performance, fiabilité et coût [211].

**Formulation de Problème d'optimisation multi-objectifs :**

Trouver le vecteur de décision idéal $\overrightarrow{s}^*$ tel que les contraintes $y_i(\overrightarrow{s})$ et $z_j(\overrightarrow{s})$, soient satisfaites et dont $\overrightarrow{F}(\overrightarrow{s}^*)$ est optimal.

$$\text{Optimiser : } \overrightarrow{F}(\overrightarrow{s}) = \{f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), ..f_k(\overrightarrow{x})\}$$

($k$ : fonctions à optimiser)

tel que : $s \in S$

$\overrightarrow{s}$ : est un vecteur de n variables de décisions.

$y_i(\overrightarrow{s})$ et $z_j(\overrightarrow{s})$ : sont les fonctions des contraintes qui déterminent le domaine X des solutions.

$S$ : représente le domaine réalisable.

$$S = \left\{ \overrightarrow{s} \;\middle|\; \begin{array}{ll} y_i(\overrightarrow{s}) \leq 0 & i = 1...q \\ z_j(\overrightarrow{s}) & j = 1...l \\ \overrightarrow{s} = (s_1...s_n)^T & \end{array} \right.$$

Généralement, ces problèmes multi-objectifs sont en concurrence, car si l'un est amélioré, cela entraîne la détérioration de l'autre. Ce conflit entre les objectifs s'explique facilement : les structures à haute performance ont tendance à avoir un coût élevé, tandis que les objectifs simples et peu coûteux ne fonctionnent généralement pas bien. Selon les limites de la spécification, la performance, et à un coût acceptable, peut être optimale. Dans les problèmes multi-objectifs, l'optimum n'est plus une valeur unique comme c'est le cas pour les problèmes à objectif unique, mais plutôt un ensemble de points, appelé front de Pareto [213].

Dans le cadre des problèmes de minimisation, le problème de maximisation peut facilement être transformé en un problème de minimisation en prenant en compte l'équivalence suivante : Maximiser $\overrightarrow{F}(\overrightarrow{x}) \Leftrightarrow$ Minimiser $-\overrightarrow{F}(\overrightarrow{x})$.

Pour résoudre des problèmes multi-objectifs, nous utilisons :

(•) Méthodes non Pareto : essayent de réduire le problème initial à un ou plusieurs problèmes à objectif unique. Parmi ces méthodes : Méthode d'agrégation par pondération ; Méthode $\varepsilon$-contraintes ; Méthode de but à atteindre ; Méthode de min-max ; Méthode de Goal programming.

(•) Les méthodes Pareto : ces méthodes ne changent pas les objectifs du problème, car elles sont traitées sans aucune discrimination lors de sa résolution. Pareto a une vision

plus globale en observant tous les critères et en utilisant le concept de dominance au sens de Pareto.

(•) Méthodes méta-heuristiques et les méthodes hybrides : les méta-heuristiques sont des méthodes souvent inspirées des systèmes naturels tels que le recuit simulé et les algorithmes évolutifs tels que : les algorithmes génétiques. L'approche hybride qui fait l'hybridation entre deux méthodes méta-heuristiques différentes. Pour plus d'informations voir le chapitre (2).

## 1.5 Problèmes classiques d'optimisation combinatoire

Cette deuxième section est basée sur les problèmes classiques d'optimisation combinatoire (problème du sac à dos, problème d'affectation, problème du voyageur de commerce, etc). Et leur relation avec d'autres problèmes du point de vue des méthodes et techniques de résolution. En raison de leurs nombreuses applications dans le monde réel, par exemple le problème d'allocation de mémoire dans les systèmes embarqués dont le modèle est un problème d'allocation généralisé (GAP) (voir le chapitre 5), et aussi modéliser et résoudre un problème de l'enchaînement de données en tant que problème de sac à dos. L'optimisation de la trajectoire des robots, la séquence du processus de fabrication par le TSP, etc.

### 1.5.1 Problème d'affectation (AP)

**Description du Problème :**

Le problème d'affectation AP, est la meilleure affectation des tâches des machines. Puisque chaque machine doit effectuer et réaliser une seule tâche à un certain coût. L'objectif est de réaliser toutes les tâches à un coût inférieur, c'est-à-dire de réduire les coûts globaux des affectations (c'est-à-dire les paires de tâches et de machines).

**Modélisation par un programme mathématique :**

Considérons $m$ comme des tâches et $n$ de machines, avec $n \geq m$. On note que $c_{i,j} \geq 0$ c'est le coût de réalisation de l'affectation de la tâche $i$ à la machine $j$ tel que ($i = 1$ à $m$, $j = 1$ à $n$). Chaque machine doit effectuer chaque tâche exactement et chaque machine

peut réaliser au plus une tâche. Alors Comment attribuer des tâches aux machines, tout en respectant les contraintes d'accomplissement des tâches et de disponibilité des machines, afin de réduire le coût total de leur réalisation [148].

À tout couple tâche/machine $(i, j)$, on associe une variable d'affectation, $x_{i,j}$, binaire,

$$x_{i,j} = \begin{cases} 1 & \text{si la tâche } i \text{ est affectée à la machine j} \\ 0 & \text{autrement} \end{cases}$$

Le problème d'affectation peut être modélisé comme suit :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

$$s.c \sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1, \cdots, m \tag{1.2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \cdots, n \tag{1.3}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \cdots, m, j = 1, \cdots, n \tag{1.4}$$

Ensuite, (1.1) est le coût total de l'exécution des tâches, (1.2) est le nombre de machines réalisant la tâche $(i)$, (1.3) est le nombre de tâches réalisées par la machine $j$.

De nombreuses applications, telles que l'allocation de ressources, présentent des contraintes de ce problème (communément appelées limitations d'allocation).

**Méthode de résolution :**

le problème d'affectation consiste également à rechercher un couplage de cardinalité $m$ de poids minimum (ou maximum) dans le graphe G, voir le problème de couplage dans un graphe biparti [120]. Le couplage ou le problème d'affectation dans un graphe biparti peut être modélisé comme un problème de flot maximal à coût minimal (les capacités des arcs =1). Des algorithmes simples et efficaces existent pour résoudre ce problème (ex : Busaker et Gowen [30]).

Le problème d'affectation peut être modélisé comme un programme linéaire, et c'est ce qu'on appelle un algorithme dual « méthode hongroise » (Cohn en 1955). Il peut

être considéré comme une variante de l'algorithme de Busaker et Gowen, Il s'agit d'un bon algorithme standard pour traiter et résoudre un problème d'affectation (voir aussi [84, 120]).

### 1.5.2 Problème d'affectation généralisé (GAP)

Ce problème est une généralisation du problème d'affectation dans lequel les tâches et les agents ont une dimension. De plus, la dimension de chaque tâche peut varier d'un agent à l'autre.

Commençons par présenter un modèle mathématique de GAP. Supposons que nous devions effectuer des tâches $n$ sur des machines $m$. Soit $b_i$ la disponibilité des ressources de la machine $i$, soit $a_{ij}$ la quantité de ressources requise par la machine $i$ pour effectuer le tâche $j$ et soit $c_{ij}$ le coût d'affectation de la tâche $j$ à la machine $i$. Chaque tâche est contrainte d'être affectée à une seule machine sans dépasser sa disponibilité en ressources. GAP consiste à trouver une affectation de coût minimum sous la contrainte ci-dessus. Un modèle IP bien connu pour GAP, voir chapitre 5. Pour plus d'informations sur ce problème et comment y apporter des solutions, consultez : [88, 91, 93].

Motivé par le problème de l'allocation mémoire dans les systèmes embarqués dont le modèle est un problème d'affectation généralisée (GAP) avec contraintes collatérales, notre contribution est de concevoir une méthode approximative suffisamment rapide et précise pour ces derniers. Pour plus d'informations et d'explications, voir le chapitre 5.

### 1.5.3 Problème du sac à dos (KP)

Beaucoup de chercheurs intéressés ont mentionné dans leurs travaux que le problème du sac à dos entre dans la catégorie des problèmes NP-hard [64, 73, 98, 99, 142, 153]. Ce problème est considéré comme l'un des meilleurs problèmes d'optimisation combinatoire qui a été largement discuté sur le plan théorique et scientifique, ainsi que les variables qui en découlent.

**Description du Problème :**

Le problème KP est un des problèmes de sélection, maximisant la norme de qualité sous une contrainte linéaire de capacité de la ressource, il est nécessaire de choisir les objets

à prendre. Afin d'être le plus grand sac possible, utile, bien sûr, en tenant compte du volume du sac à dos (capacité de la ressource).

Ce problème n'a qu'une seule contrainte sur les objets de la solution.

**Modélisation par un programme mathématique :**

Soit un ensemble de $n$ éléments et $b$ est une ressource disponible en quantité limitée. Chacun des $p_j$ et $a_j$ prend des valeurs positives quelque soit $j = 1$ à $n$. Considérons $p_j$ le profit associé à la sélection de l'élément $j$ et considérons également $a_j$ la quantité de ressource que l'élément $j$ requiert, s'il est sélectionné. Le problème du sac à dos est généralement un sous-ensemble de $n$ éléments choisis, ces éléments augmentent le profit total réalisé, bien sûr en tenant compte de la quantité de ressources disponibles [148].

On associe à chaque élément $j$ une variable de sélection, $x_j$ , binaire,

$$x_j = \begin{cases} 1 & \text{si } j \text{ est sélectionné} \\ 0 & \text{autrement} \end{cases}$$

Le bénéfice total obtenu s'écrit comme suit : $\sum_{j=1}^{n} p_j x_j$, et la quantité totale de ressources utilisées (appelée la "contrainte du sac à dos") s'écrit $\sum_{j=1}^{n} a_j x_j$.

Le problème du sac à dos est modélisé sous la forme :

$$\begin{aligned} & \max \sum_{j=1}^{n} p_j x_j \\ s.c \quad & \sum_{j=1}^{n} a_j x_j \leq b \\ & x_j j = 1, \cdots, n \end{aligned}$$

**Méthode de résolution :**

Divers travaux proposant des méthodes exactes de résolution du problème du sac à dos, comme dans Martello *et al.* [134]. Il existe trois grands types d'algorithmes proposés. Premièrement, les algorithmes de Branch and Bound. Deuxièmement, les algorithmes

basés sur l'identification d'une variable critique et d'un sous-ensemble associé de variables critiques. Troisièmement, les algorithmes efficaces de programmation dynamique mentionnés dans le travail de recherche [133].

Le problème du sac à dos "multidimensionnel", ce problème se pose lorsqu'il y a plusieurs contraintes, par exemple prendre le poids que le sac peut supporter avec prendre la taille maximale du sac et ainsi de suite. Il existe un grand nombre de recherches qui ont abordé ce problème à grande échelle, notamment les suivantes : [98, 99, 142, 153].

### 1.5.4 Problème du voyageur de commerce (TSP)

**Description du Problème :**

Le problème TSP peut se définir de la façon suivante : considérons un certain nombre de villes $n$ qui doivent être visitées par un vendeur voyageur, une seule fois, en arrivant une fois et en quittant une fois et en commençant et en terminant dans la même ville. Compte tenu des distances par paire entre les villes, quel est le meilleur ordre pour les visiter, de façon à minimiser la distance totale parcourue ?. Ce problème a été largement étudié en optimisation combinatoire.

Ses applications résident dans l'expression des problèmes de transport, bien que les problèmes de transport soient plus difficiles que les TSP mais ont une structure sous-jacente de type TSP, et aussi dans l'optimisation de la trajectoire des robots, de la séquence du processus, ect.

**Modélisation par un programme mathématique :**

Soit G = $(X, U)$, un graphe où $X$ l'ensemble des sommets et $U$ l'ensemble des arcs. sachant que les sommets représentent les villes à visiter plus la ville de départ de la tournée, et que les arcs représentent les parcours possibles entre les villes. La distance $d_{i,j}$ est la distance du parcours (ville $i$ à la ville $j$) à tout arc $(i, j) \in U$. Dans G, la somme des distances associées aux arcs de ce chemin, est appelée la longueur d'un chemin. Le TSP se fait à la recherche d'un circuit hamiltonien de longueur minimale dans G. On parle de TSP asymétrique s'il y a quelques arcs $\in U$ pour lesquels $d_{i,j} \neq d_{j,i}$ [148].

Pour chaque couple $(i, j)$ de villes à visiter ($i = 1$ à $n$, $j = 1$ à $n$ et $i \neq j$) une distance $\delta_{i,j}$ ($d_{i,j}$ s'il existe un moyen d'aller directement de $i$ à $j$), fixée à $\infty$, sinon est une variable

de succession, $x_{i,j}$ , binaire,

$$x_{i,j} = \begin{cases} 1 & \text{si la ville } j \text{ est visitée immédiatement après la ville } i. \text{ dans la tournée} \\ 0 & \text{sinon.} \end{cases}$$

Le TSP est alors modélisé comme suite :

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij} x_{ij} \tag{1.5}$$

$$s.c \sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1, \cdots, n \tag{1.6}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad j = 1, \cdots, n \tag{1.7}$$

$$\sum_{i \in s, j \notin s}^{n} x_{ij} \geq 2 \qquad S \subset X, S \neq \emptyset \tag{1.8}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \cdots, n, j = 1, \cdots, n \tag{1.9}$$

Les contraintes (1.6) et (1.7) notent que chaque ville doit être visitée exactement une fois. La contrainte (1.8) appelée contrainte d'élimination des sous-tours.

**Méthode de résolution :**

Une formulation d'un modèle mathématique basé sur la programmation linéaire en nombres entiers (PLNE) pour la TSP. Pour résoudre la TSP de manière exacte, des algorithmes ont été développés, par exemple l'algorithme de branchement et cut et les techniques de propagation des contraintes dans l'arbre de recherche, ainsi que des algorithmes de programmation dynamique utilisés pour trouver des circuits hamiltoniens dans un graphe G, la programmation par contraintes (PPC). En outre, il existe des procédures heuristiques et des techniques d'optimisation locale efficaces, ainsi que des calculs de bornes inférieures, basés par exemple sur la relaxation lagrangienne de la TSP, afin de fournir un bon cadre pour l'optimum. Pour plus informations voir [95].

Le problème du voyageur de commerce multiple (MTSP), ce problème est similaire au problème connu du "TSP", avec la complication supplémentaire que chaque ville peut être visitée par n'importe lequel des vendeurs [35].

Il existe de nombreuses variantes de la TSP, obtenues soit par ajouts de contraintes comme (TSPTW) dans lequel la visite, soit par modification, comme un problème de tournées de véhicules (VRP). Pour une présentation détaillée des différentes variantes de la TSP, ainsi que des méthodes de résolution existantes, on peut se référer à cet ouvrage spécialisé [86].

### 1.5.5 Problème d'ordonnancement (SP)

**Description du Problème :**

Le problème d'ordonnancement est défini comme la planification d'une réalisation qui peut être divisée en tâches dans le temps, en tenant compte des contraintes temporelles liées à la disponibilité des ressources requises par les tâches, ainsi qu'en tenant compte des contraintes de temps, qui peuvent être résumées en spécifiant les dates de début de réalisation, en tenant compte de contraintes spécifiques. Dans un environnement d'optimisation, le travail consiste à minimiser (ou maximiser) un paramètre, tel que la durée totale des tâches ou la minimisation du Makespan. Le terme "Makespan" apparaît généralement dans le contexte de l'ordonnancement [1].

les tâches et les contraintes potentielles ainsi que les ressources et la fonction économique sont les différentes données du problème d'ordonnancement.

Contrairement à ce que nous avons vu précédemment dans les trois problèmes, le problème d'ordonnancement n'est pas tout à fait spécifique, puisqu'il n'a pas de formule mathématique. Le problème d'ordonnancement fait référence à un ensemble de problèmes qui existent dans la réalité. Chacun de ces problèmes peut être identifié à travers les données des tâches (ces éléments ne sont pas très divers), par exemple, les différentes ressources qu'elles constituent habituellement. Ces tâches doivent être programmées pour atteindre un objectif d'optimisation spécifique, dépendant de la minimisation de la durée globale, du respect des dates de commande ou de la minimisation d'un coût [33, 180].

Dans ce contexte, on parle de problème d'ordonnancement *discret* lorsqu'une ressource n'est pas partageable, ce qui implique qu'elle ne peut pas être capable de réaliser plusieurs activités en même temps. Le contraire conduit à un problème d'ordonnancement *cumulatif*. Si la ressource est disponible et en même quantité à nouveau à condition

qu'elle ait été utilisée dans de nombreuses tâches, on parlera ici de ressource renouvelable. La quantité de ressources utilisables à tout moment est limitée. La consommation globale dans le temps est limitée, si la réalisation d'une tâche la réduit d'une certaine quantité, alors la ressource est au contraire " consommable " (matières premières, budget, etc.). Une ressource est "doublement contrainte" si son utilisation immédiate et sa consommation globale sont limitées (l'exemple le plus courant est le financement d'un projet). En ce qui concerne les tâches, on peut parler de problème préemptif et non préemptif. Pour plus de détails sur ces concepts et pour présenter les différents critères de classification sur les problèmes d'ordonnancement, voir [25, 130].

**Méthode de résolution :**

Une analyse des différentes formules mathématiques proposées dans la littérature est présentée dans van den Akker [218]. Pour modéliser le problème d'ordonnancement, nous pouvons travailler avec plusieurs types de variables de décision. Par exemple, s'il existe un problème d'ordonnancement distinct et non préventif pour une seule machine.

Les modèles linéaires peuvent être utilisés pour un grand nombre de problèmes d'ordonnancement (relaxation linéaire). Mais, les techniques purement linéaires ne fonctionnent pas bien sur les problèmes d'ordonnancement. Le plus souvent, il est basé sur l'utilisation de techniques avancées de propagation de contraintes, voir [120].

### 1.5.6 Problème de tournées de véhicules (VRP)

Le problème de tournées de véhicules (Vehicle Routing Problem ou VRP) est clairement NP-difficile, car il généralise le TSP déjà NP-difficile. C'est l'un des problèmes d'optimisation combinatoire les plus étudiés en raison de la difficulté de son utilisation dans divers domaines. Récemment, VRP a été utilisé pour modéliser certains problèmes dans le domaine des réseaux, en particulier le routage Internet (l'acheminement des données).

**Description du Problème :**

Le VRP consiste à tenter de trouver une méthode idéale pour qu'un groupe de véhicules situés dans un ou plusieurs entrepôts puisse livrer des produits à un groupe spécifique de clients [150]. Le véhicule quitte l'entrepôt pour rendre visite aux clients, à condition que le total des demandes des clients ne dépasse pas la capacité de véhicules disponibles

$Q$. Avant de retourner à l'entrepôt, le véhicule doit satisfaire un service pour chaque client, un véhicule qui répond totalement à leur demande.

**Modélisation par un programme mathématique :**

Dans le travail de Laporte and Nobert [122], ont été proposées les formules les plus utilisées pour VRP, qui ont généralisé TSP [50]. Ils définirent donc un graphe G =$[V, E, C]$, où $V$ est l'ensemble des nœuds incluant le dépôt $O$ et $V\prime$ représente l'ensemble de clients où $V\prime = V\backslash\{O\} = \{1, ..n\}$. $E$ est l'ensemble des arêtes. Chaque client $i \in v\prime$ a une demande $q_i$ et $Q$ est la capacité des $K$ véhicules disponibles telle que $\sum_{i \in V\prime} q_i \leq KQ$.
$x_e$ c'est une variable qui compte le nombre d'utilisations des arêtes $e$ de coût $c_e$.
Soit $\delta(S)$ l'ensemble des arêtes dont une seule des extrémités est dans $S$.

Donc on peut modéliser le VRP comme suit :

$$\min \sum_{e \in E} c_e x_e \tag{1.10}$$

$$s.c \sum_{e \in \delta([i])} x_e = 2 \qquad i \in V\backslash\{O\}, \tag{1.11}$$

$$\sum_{e \in \delta([O])} x_e \leq 2K \tag{1.12}$$

$$\sum_{e \in \delta([S])} x_e \geq 2r(S) \qquad \forall S \subset V\backslash\{O\}, S \neq \emptyset, \tag{1.13}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E\backslash\delta(\{O\}) \tag{1.14}$$

$$x_e \in \{0, 1, 2\} \qquad \forall e \in \delta(\{O\}) \tag{1.15}$$

Le rôle de la fonction objectif (1.10) est de réduire le coût général de la solution, qui est égal à la somme de coût des arêtes utilisées. Les contraintes (1.11) définissent le besoin de deux arêtes incidentes pour chaque client afin que le véhicule puisse s'y rendre et en repartir. les (1.12) garantissent qu'un maximum de $2K$ d'arêtes dépôt afin que les $K$ véhicules puissent, la plupart du temps, partir puis revenir au dépôt. pour ces contraintes sous tours (1.13), les véhicules nécessaires pour satisfaire la demande des clients contenus dans $S$ , et aussi pour empêcher la construction de cycles non-connectés au dépôt et la construction de cycles dont la demande dépasse la capacité $Q$. les contraintes (1.14) et (1.15) sont des contraintes d'intégrité.

**Méthode de résolution :**

De nombreux ouvrages ont abordé le problème VRP de manière approfondie, comme le livre de Toth and Vigo [214].

Le VRP a été utilisé récemment pour modéliser certains problèmes. Les situations les plus difficiles ne peuvent être résolues de manière optimale. Il existe parfois des solutions de bonne qualité. Afin de trouver des solutions raisonnables dans des temps de calcul acceptables. Parmi les méthodes de résolution, il existe en général les méthodes exactes récentes Letchford *et al.* [125], et aussi de nombreuses méta-heuristiques comme l'algorithme mémétique [172], les algorithmes de recherche tabou ont également été utilisés avec succès [215] et plus récemment les algorithmes de colonies de fourmis [176].

De nombreuses variantes du VRP existent et ont été étudiées dans la littérature. Nous pouvons mentionner par exemple le VRP avec collectes et livraisons (Savelsbergh and Sol [186]), le VRP avec fenêtre horaire (Desrochers et al. Desrochers *et al.* [56]), le VRP avec livraisons divisibles ([10]), ou encore le VRP multi-dépôts ([177]).

## 1.6 Conclusion

Dans ce chapitre, nous avons expliqué l'optimisation combinatoire, la théorie de la complexité, et introduit quelques problèmes d'optimisation standards et classiques avec leur complexité. En général, l'optimisation combinatoire établit un schéma formel pour de nombreux problèmes dans l'industrie, l'économie, la biologie et d'autres domaines, il est donc utile de modéliser ces problèmes comme un problème de minimisation ou de maximisation selon une fonction objectif et des contraintes. Le défi reste de trouver des solutions efficaces, c'est-à-dire des solutions optimales ou quasi-optimales aux problèmes d'optimisation, et il est alors nécessaire de développer et d'améliorer des méthodes mathématiques précises dépendant du problème donné. Dans le chapitre suivant, nous allons présenter une description des méthodes de résolution des problèmes d'optimisation combinatoire.

# Chapitre 2

# Description des méthodes de résolution des problèmes d'optimisation combinatoire

## 2.1 Introduction

La particularité des problèmes d'optimisation combinatoire se situe dans le facteur temps de la recherche d'une solution optimale, quelques fois il n'est pas possible de trouver une seule solution acceptable. Pour estimer le temps de recherche, il faut appliquer la théorie de la complexité et ses techniques.

On peut résoudre certains problèmes d'optimisation combinatoire en temps polynomial, ce qui peut se faire par une formulation linéaire en variables réelles, ou en utilisant à la fois un algorithme glouton ou un algorithme de programmation dynamique.

En effet, dans la plupart des cas, le problème peut être très difficile, et pour obtenir une solution, on peut recourir à des algorithmes de Branch and Bound, ou à l'optimisation linéaire correcte, ou bien encore à la programmation par contraintes.

En pratique, la complexité pratiquement admissible n'est généralement qu'un polynôme. S'il est alors possible d'accepter au mieux l'existence d'une solution proche, il sera obtenu par une heuristique ou une méta-heuristique

Il existe deux méthodes pour résoudre les POC, à savoir les méthodes exactes et méthodes approchées, nous résumons les deux dans la figure suivante 2.1.

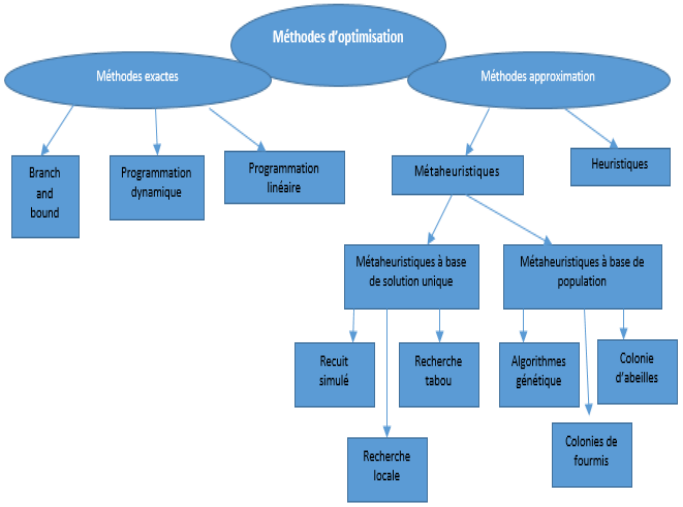


FIGURE 2.1 – Classification des méthodes pratiques pour résoudre les problèmes d'optimisation combinatoire.

Ce chapitre introduit les concepts de base nécessaires à la compréhension de notre proposition dans le chapitre (5).

## 2.2 Méthodes exactes

Les méthodes exactes parcourent tout l'espace de recherche. Ainsi, une solution optimale peut être atteinte à condition qu'aucune restriction de temps ne soit donnée.

Cependant, le temps de calcul nécessaire pour obtenir rapidement une solution idéale peut devenir inacceptable malgré les différentes méthodes heuristiques développées pour accélérer la recherche de solutions.

Les méthodes exactes peuvent obtenir des solutions optimales aux problèmes de petites tailles en termes de programmation linéaire en nombres entiers. Le temps de calcul requis pour arriver à une solution peut croître de manière importante avec la taille du problème.

Il existe plusieurs méthodes de résolution exacte qui se caractérisent par la possibilité de trouver une ou plusieurs solutions dont l'optimalité est garantie.

### 2.2.1 Relaxation lagrangienne et méthode de sous-gradients

La relaxation de Lagrange, est une technique connue en optimisation, a été présentée avec les travaux de Held and Karp [97] au cours du début des années 1970 au sujet du problème du voyageur de commerce.

les étapes d'application de la relaxation lagrangienne est :

- Relaxer à aide des multiplicateurs de Lagrange les contraintes qui rendent le problème plus difficile à résoudre et insérer dans la fonction objectif du problème (S) le coût de la pénalisation résultant de la relaxation.
- Pour chaque multiplicateur de Lagrange, résolvez le problème relaxé, afin de trouver un minimum.
- Pour trouver une borne supérieure, trouver une solution faisable (solution admissible) du problème (S).
- Utiliser la méthode du sous gradient pour résout le problème dual.
  L'efficacité de la méthode de relaxation lagrangienne est assurée par :
  - Des excellentes bornes inférieures.

— Qualité des solutions acceptables au problème initial (S) obtenues à partir des solutions du problème de relaxation.

Beaucoup de travaux ont utilisé la technique de la relaxation lagrangienne, à savoir : [18, 70, 71, 78, 149, 194].

Pour une explication de l'application de cette technique à un problème d'allocation de mémoire dans les systèmes embarqués, voir le chapitre 5.

#### 2.2.1.1 Problèmes primal et dual

Le dual est un autre problème que nous écrirons séparément de notre problème original (primal). Cette écriture concernant à chaque PO linéaire.

On a un problème d'optimisation linéaire avec $n$ variables et $m$ contraintes.
Soit $\mathbf{A} \in \mathbb{R}^{m \times n}$ avec les vecteurs $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$ et $\mathbf{b} \in \mathbb{R}^m$.

**Problème de programmation linéaire forme standard**

| | **Problème primal** | | | **Problème dual** | |
|---|---|---|---|---|---|
| $min$ | $\mathbf{z} = \mathbf{c}^t\mathbf{x}$ | | $max$ | $\mathbf{z} = \mathbf{b}^t\mathbf{y}$ | |
| $s.c$ | $\mathbf{Ax}$ | $= \mathbf{b}$ | $s.c$ | $\mathbf{A^t y}$ | $\leq \mathbf{c}$ |
| | $\mathbf{x}$ | $\geq 0$ | | | |

**Problème de programmation linéaire avec inégalité :**

| | **Problème primal** | | | **Problème dual** | |
|---|---|---|---|---|---|
| $min$ | $\mathbf{z} = \mathbf{c}^t\mathbf{x}$ | | $max$ | $\mathbf{z} = \mathbf{b}^t\mathbf{y}$ | |
| $s.c$ | $\mathbf{Ax}$ | $\leq \mathbf{b}$ | $s.c$ | $\mathbf{A^t y}$ | $\leq \mathbf{c}$ |
| | $\mathbf{x}$ | $\geq 0$ | | $\mathbf{y}$ | $\geq 0$ |

En résumé, pour chaque problème primal, on associe un problème dual, on échange les variables et les contraintes de sorte que chaque variable du primal corresponde à une contrainte du dual et que chaque contrainte du primal corresponde à une variable du dual, on conclut que le problème dual est le problème primal.

Généralement, en programmation linéaire, les conditions optimales sont nécessaires et satisfaisantes. Alors que la programmation non linéaire nécessite une description des contraintes, ce qui conduit à des conditions ajoutées sur l'ensemble des solutions possibles afin de caractériser les solutions locales idéales.

#### 2.2.1.2 Problème dual lagrangien (DL)

La dualisation d'un problème a pour avantage d'obtenir un problème dual qui sera plus facile à résoudre. Peut-être que cette solution duale nous permet de trouver la solution primale ! Lorsque l'on dualise un problème (appelé primal), on dispose de borne supérieure et borne inférieure de la fonction dite de couplage (appelée Lagrangien).

Le principe de la relaxation lagrangienne est une méthode de relaxation qui permet de supprimer les contraintes difficiles en les intégrant dans la fonction objectif en la pénalisant si ces contraintes ne sont pas respectées, on dit que ces contraintes sont dualisées. On associe pour cela à chaque contrainte un multiplicateur de Lagrange ou variable duale et en définissant la fonction de Lagrange (ou problème relaxé).

Soit le problème (P) avec de contraintes ($D \in \mathbb{R}^{m_D \times n}$ et $S \in \mathbb{R}^{m_S \times n}$),

$$(\mathrm{P}) \begin{cases} \min & \mathbf{z} & = \mathbf{cx} & \\ \text{s.t} & \mathbf{Dx} & \geq \mathbf{d} & \text{k contraintes " difficiles"} \\ & \mathbf{Sx} & \geq \mathbf{b} & \text{m contraintes "faciles"} \\ & \mathbf{x} & \in \mathbb{N}^n & \end{cases} \tag{2.1}$$

Pour résoudre le problème (P), la relaxation de Lagrange consiste à relâcher la contrainte ($\mathbf{Dx} \geq \mathbf{d}$) en introduisant le multiplicateur de Lagrange $\lambda = (\lambda_i), i \in I$. C'est à dire on multiplie la contrainte relâchée ($\mathbf{Dx} \geq \mathbf{d}$) par $\lambda$ et puis on introduit la fonction résultante dans la fonction objectif de (P).

Nous obtenons ainsi le problème relaxé :

$$(\mathrm{Q}) \begin{cases} \min & \mathbf{z} & = \mathbf{cx} + \lambda(\mathbf{d} - \mathbf{Dx}) \\ \text{s.t} & \mathbf{Sx} & \geq \mathbf{b} \\ & \mathbf{x} & \in \mathbb{N}^n \end{cases} \tag{2.2}$$

Sachant que $\lambda$ impose une pénalité aux contraintes violées. Pour tout $\lambda \in \mathbb{R}_{\geq 0}^{m_S}$, le DL donne une borne inférieure à la solution optimale du problème fondamental (P). La valeur objectif optimale d'un problème DL est une borne inférieure pour la valeur objectif optimale du problème primal (P), Beasley [18] démontre ce fait comme suit :

La valeur objectif optimale $z^*$ de (P) est supérieure à la valeur objectif optimale de :

$$\begin{array}{lll} \min & \mathbf{z} & = \mathbf{cx} + \lambda(\mathbf{d} - \mathbf{Dx}) \\ \text{s.t} & \mathbf{Dx} & \geq \mathbf{d} \\ & \mathbf{Sx} & \geq \mathbf{b} \\ & \mathbf{x} & \in \mathbb{N}^n \end{array}$$

Puisque $\lambda \geq 0$, et $(d - Dx) \leq 0$, un terme qui est $\leq 0$ et simplement ajouté à la fonction objectif.

Ceci n'est pas inférieur que la valeur objectif optimale de :

$$\begin{array}{lll} \min & \mathbf{z} & = \mathbf{cx} + \lambda(\mathbf{b} - \mathbf{Sx}) \\ \text{s.t} & \mathbf{Dx} & \geq \mathbf{d} \\ & \mathbf{x} & \in \mathbb{N}^n \end{array}$$

Alors que la suppression d'un ensemble de contraintes au problème de minimisation ne peut que réduire la valeur de la fonction objectif.

Une bonne application d'une bonne relaxation lagrangienne nécessite d'abord de trouver la contrainte complexe à relâcher pour que la résolution du problème relaxé soit plus facile que la résolution du problème initial, et ensuite de retrouver les multiples lagrangiens, c'est-à-dire de trouver de bonnes valeurs numériques qui sont à la borne inférieure supérieure. Il s'agit de trouver les multiples qui correspondent à :

$$\max \lambda \geq 0 \left\{ \begin{array}{lll} \min & \mathbf{z} & = \mathbf{cx} + \lambda(\mathbf{b} - \mathbf{Sx}) \\ \text{s.t} & \mathbf{Dx} & \geq \mathbf{d} \\ & \mathbf{x} & \in \mathbb{N}^n \end{array} \right\}$$

Un algorithme heuristique appelé sous-gradient (SG), qui implique de nombreuses minimisations du problème relaxé. Cette heuristique permet de trouver de bonnes valeurs pour les multiplicateurs de Lagrange $\lambda^*$.

#### 2.2.1.3 Résolution du problème dual par l'algorithme du sous-gradient

la méthode du sous-gradient est une généralisation naturelle des méthodes du gradient en remplaçant le gradient par un sous-gradient arbitraire. La méthode d'optimisation par SG est une procédure itérative qui peut être utilisée pour résoudre un problème lagrangien dual exact afin de relaxer les contraintes difficiles (($\mathbf{Dx} \geq \mathbf{d}$)).

Supposons que (P) soit un programme linéaire en nombre entier comme dans le 2.1. On veut relâcher la contrainte ($\mathbf{Dx} \geq \mathbf{d}$). Il en résulte un problème DL 2.2 (Q). L'optimisation du sous-gradient proposé par Beasley [18] peut être décrite comme suit :

1. Choisissez un ensemble initial $\lambda_i$ de multiplicateurs : $\lambda_i = 0 \quad \forall \quad 1 \leq i \leq m_s$.
   Ce qui donne une initialisation de borne supérieure $Z_{UB}$ .
2. Poser $Z_{LB} = c.x^* + \lambda(d - D.x^*)$ s'accorde à la fonction objectif du problème dual. On définit le sous-gradient $\delta_i \in \mathbb{R}^{m_s}$ par :

$$\delta_i = (d - D.x^*)$$

3. En utilisant cette égalité

$$\lambda_i = max(0, \lambda_i + \Delta.\delta_i) \quad \forall 1 \leq i \leq ms$$

   avec $\Delta$ une taille définit par :

$$\Delta = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{=1}^{m_s} \delta_i^2}$$

   et $\pi$ donné par l'utilisateur dans l'intervalle $[0, 2]$. En commençant avec $\pi_0 = 2$ puis en le diviser par 2 après $\beta$ itérations, puis $\lceil \beta/2 \rceil$ itérations, etc.

   Donc, on construit une séquence de multiplicateurs pour passer à l'étape 2. Ainsi, pour terminer l'algorithme, nous fixons une base d'arrêt ou une condition basé sur la valeur de $\pi$, ou définissons le nombre d'itérations possible.

Pour terminer, nous prenons la valeur maximale du DL comme la meilleure approximation du problème primaire.

L'optimisation de SG est une heuristique pour résoudre le problème du DL. Les multiples de Lagrange sont fréquemment ajustés pour trouver des valeurs qui produisent le minimum ou le meilleur. Cette méthode est basée sur la solution du problème DL, et sur une borne supérieure de $Z_{UB}$ de la valeur de la fonction objectif optimale du problème originale [173].

### 2.2.2 Méthode de séparation et évaluation "Branch et Bound (B&B)"

La méthode (B&B) a été présentée en premier par Land and Doig [121], qui l'avait description comme une manière de construire un arbre de décision. Elle résout les problèmes de nombres entiers dans de plusieurs logiciels commerciaux.

Dans les méthodes de type (B&B), la solution de nombreux problèmes relaxés, résultant de la division de l'ensemble réalisable en différents sous-ensembles, conduit à la solution optimale. Dans cette référence (Lawler and Wood [123]), vous trouverez plus d'informations sur cette méthode, et aussi les techniques essentielles à l'optimisation par méthode contrainte sont décrites et plusieurs applications spécifiques sont passées en revue et celles-ci incluent la programmation linéaire et non linéaire.

Un sous-ensemble de solutions à résoudre est évalué par la relaxation qui permet de trouver plus rapidement la solution optimale, soit par relaxation continue, soit par relaxation de Lagrange. Ce principe est basé sur l'élimination des sous-ensembles de solutions à résoudre pour lesquels la relaxation est moins favorable que les meilleures solutions possibles, et ainsi de suite Jusqu'à ce qu'il n'y ait que des problèmes faciles à résoudre.

La méthode (B&B) est basée sur deux approches : (1) l'arbre d'énumération et (2) les bornes duales définies par l'arbre de l'énumération.

Fondamentalement, il y a deux étapes [143] :

— La Séparation : cette mesure consiste à diviser et maîtriser le groupe de toutes les solutions possibles au problème de base en sous-ensembles plus petits et en sous-groupes incompatibles entre eux. La séparation consiste à énumérer les solutions,

tandis que l'évaluation consiste à éviter l'énumération systématique de toutes les solutions. Si le nombre global de solutions possibles pour le problème à résoudre est faible, on peut examiner chaque solution possible individuellement et sélectionner l'optimum en les mesurant les unes par rapport aux autres.

— L'évaluation : cette étape consiste à utiliser des critères pour déterminer quels sous-groupes peuvent contenir la solution optimale et quels sous-groupes ne doivent pas être explorés davantage parce qu'ils ne contiennent pas la solution optimale.

La méthode (B&B) permet d'effectuer une énumération intelligente de l'espace des solutions, elle élimine les solutions partielles qui ne nous donnent pas la solution optimale. Pour ce faire, cette méthode contient une fonction qui permet de placer un plancher (dans le cas d'un minimum) sur certaines solutions à éliminer ou à garder comme solutions possibles. Bien évidemment, les performances de la B&B, entre autres, de la qualité de cette fonction (sa capacité à éliminer précocement les solutions incomplètes).

### 2.2.3 Méthode des coupes de Gomory

Cette méthode a été développée par Ralph E. Gomory (1958). Elle propose une démarche pour résoudre le PLNE. La base essentielle est d'ajouter des contraintes qui n'excluent pas entièrement tout point entier possible.

Nous pouvons résumer la méthode comme suit [83] :

Afin de terminer la solution de relaxation optimale, la méthode consiste à ajouter des contraintes linéaires, à couper ou à ajuster une par une, jusqu'à ce que la solution de relaxation optimale soit complète.

— En accordant PLNE, la solution PLC (programme logique avec contraintes) correspondante utilisant Simplex. Une solution est complète si la solution optimale obtenue ne contient que des valeurs entières.

— La solution optimale pour PLC, si une ou plusieurs des variables primaires ne sont pas des entiers, il faut alors créer une contrainte supplémentaire appelée la contrainte de la coupe de Gomory.

— En utilisant le simplex dual et après avoir ajouté la contrainte de la coupe de Gomory, nous déterminons une nouvelle solution. Le processus est répété jusqu'à ce qu'une solution optimale complète soit trouvée.

Il y a aussi d'autres méthodes plus ou moins sophistiquées de construction de séparations qui permettent d'obtenir plus ou moins rapidement la solution optimale de PLNE. Puisque PLNE ne contient que deux variables, nous pouvons tracer le graphe des contraintes et ensuite tracer le Gomory sur le graphe.

### 2.2.4 Méthode de programmation dynamique

La programmation dynamique est une méthode précise pour résoudre les problèmes d'optimisation, principalement grâce à Richard BELLMAN (1957). Malgré sa forte résistance, son champ d'application est relativement limité.

Elle est basée sur le principe de Richard BELLMAN (1949) [188]. La programmation dynamique consiste à placer le problème dans un groupe de problèmes ayant les mêmes caractéristiques, mais des difficultés différentes, puis à trouver une relation répétitive qui relie les solutions optimales à ces problèmes.

L'idée de base est d'analyser le problème fondamental en une collection de sous-problèmes plus petits et imbriqués, pourvu qu'ils soient de la même nature que le problème fondamental. Ensuite, on résout les sous-problèmes de manière séquentielle en commençant par des solutions plus petites des problèmes d'étape spécifiques obtenues à partir des problèmes d'étape précédents. Le principe est d'éviter de résoudre même sous le problème.

Un tableau est spécifié et doit être complété, chaque élément de ce tableau correspondant à la solution d'un seul problème intermédiaire. Cet algorithme peut être exprimé par itératif ou récursif.

Le principal consiste à exprimer la solution d'un problème en fonction des "petits" problèmes (formule d'itération). Si l'on sait que l'on doit recalculer la résolution des mêmes problèmes plusieurs fois, on est dans le domaine de la programmation dynamique. Il est possible que le nombre de sous-problèmes soit très grand et que l'algorithme obtenu ne soit pas forcément polynomial.

Pour plus d'informations sur cette méthode, consultez Bellman [19], Dreyfus [65].

## 2.3 Méthodes approchés

Parmi ces méthodes, on trouve les heuristiques qui ciblent un problème spécifique, et les méta-heuristiques plus solides et adaptables pour résoudre un grand problème.

Les méthodes heuristiques sont en particulier utiles pour les problèmes demandant une solution en temps réel (ou très courte) ou pour la résolution de problèmes difficiles sur de grandes situations de larges instances numériques. Elle peut également être utilisée pour configurer une méthode spécifique [53]. Cependant, pour que l'heuristique soit efficace à un niveau suffisant dans un problème donné, il faudra une assez bonne adaptation.

Les méta-heuristiques, autrement dit les méthodes d'optimisation conçues en fonction des normes prévues dans un contexte méta-heuristique, sont toujours de nature heuristique. Ce fait les diffère des méthodes exactes, qui sont associées à une garantie que la solution optimale sera obtenue en un temps fini (bien que souvent infiniment grand). Les méta-heuristiques sont donc conçues spécialement pour obtenir une solution "suffisamment bonne" aux problèmes NP-difficile, en un temps de calcul "suffisamment court".

les méthodes approchées ont été classées en différentes sections Mounir [143] :

— Les méthodes constructives (algorithmes gloutons, méthode pilote, GRASP).

— La recherche locale (algorithmes de descente, multi-départs, recuit simulé, algorithme à seuil, recherche tabou, méthode de bruitage).

— Les méthodes évolutionnistes (algorithmes génétiques, algorithmes d'évolution, recherche dispersée, méthode des chemins, systèmes de fourmis).

— Les Réseaux de neurones (modèle de hopfield-tank, machine de Boltzmann, réseau auto-adaptatif, réseau élastique).

— Heuristiques Bayésiennes (optimisation globale, optimisation discrète).

— Superposition (perturbation des données, perturbation des paramètres d'une heuristique).

Par les méthodes d'approche, on peut obtenir une excellente solution $s^*$ (c'est-à-dire très proche de l'optimum) dans un contexte de ressources limitées (temps de calcul et/ou mémoire). Mais, une solution optimale ne sera pas garantie. Cependant, le temps

nécessaire à l'obtention de cette solution $s^*$ sera beaucoup moins important et il pourra être fixé.

Les heuristiques et méta-heuristiques sont un compromis entre le temps de résolution et les caractéristiques des résultats.

### 2.3.1 Heuristiques classique

Les heuristiques classiques fournissent une solution au problème de l'optimisation combinatoire sans garantir une solution optimale, mais elles fournissent une solution dans un délai raisonnable. Il existe une extension pour la construction et la mise à niveau.

En général, il existe des heuristiques créées pour un problème précis sur la base de leur spécificité. On retrouve généralement les principes de base qui ont été utilisés dans ces heuristiques selon Sbihi [188] : Principe glouton, Principe de partitionnement, Principe de construction progressive.

#### 2.3.1.1 Algorithme Glouton (Greedy algorithms)

L'algorithme Glouton, également appelé algorithme du plus proche voisin. Il s'agit le plus souvent d'une méthode heuristique ou exacte, simple d'utilisation de manière générale. Son fonctionnement est le suivant : à chaque étape du processus de recherche, la solution optimale est sélectionnée localement jusqu'à ce qu'une solution (exacte ou non) soit obtenue.

Si l'algorithme ne donne pas la solution optimale dans certains cas et systématiquement, on parle alors d'heuristique gloutonne, si les choix qui sont faits au cours du processus de recherche ne sont pas remis en cause.

De nombreux algorithmes exacts importants sont dérivés de cette méthode, tels que les algorithmes de Dijkstra et les algorithmes de Kruskal.

L'algorithme Glouton a une complexité polynomiale, mais il reste à prouver et comment il est nécessaire que l'algorithme fournisse la solution optimale. La difficulté réside souvent dans ce type d'algorithme. À titre indicatif, la méthode Glouton est souvent utilisée pour trouver une première solution à d'autres méthodes plus élaborées.

Dans la référence DeVore and Temlyakov [58], quelques notes sur l'algorithme Glutton ont été expliquées.

#### 2.3.1.2 Méthode de la descente (Descent method)

La méthode de descente (DM) est une méthode classique, plus simple et assez ancienne, et doit son succès à et sa simplicité [162, 164].

À chaque phase de la recherche, cette méthode avance vers une solution voisine de bonne et meilleure qualité. Le principe consiste (voir 2.1), est à partir d'une solution initiale, à en choisir à chaque itération un point dans le voisinage de la solution courante qui améliore fortement la fonction objectif. À partir d'une solution initiale $s$, une solution voisine $s'$ est choisie, puis $s$ sera remplacée par $s'$ si $f(s') \leq f(s)$, ce processus est répété jusqu'à ce que toutes les solutions voisines soient pires que la solution courante, c'est-à-dire, qu'un optimum local est obtenu.

**Algorithme 2.1** Pseudo-code de base pour DM

1. choisir une solution $s \in S$ ;
2. déterminer une solution $s'$ qui minimise $f$ dans $N(s)$ ;
3. **si** $f(s') \leq f(s)$ **alors** poser $s \longleftarrow s'$ et retourner à 2., **sinon STOP**.

Les méthodes de descente peuvent être différenciées selon la manière de générer la solution de départ et la voie de voisinage. On peut distinguer plusieurs types de descente en fonction de la stratégie de génération de la solution de départ et de la voie de voisinage : la descente déterministe, la descente stochastique et la descente vers le premier meilleur.

La figure 2.2 schématise le déroulement de l'effet du choix de la solution initiale sur le résultat de la recherche par la méthode de la descente. Le résultat donné sera le même pour une espèce voisine donnée et la même solution initiale : le meilleur local (qui peut être l'optimum global). Le déterminisme de la méthode peut être brisé dans le cas de différences basées sur la modification du principe de choix du voisin. Le choix des voisins peut être aléatoire ou basé sur une exploration incomplète du voisinage (premier voisin améliorant). D'autre part, le choix de la solution initiale limite la régression à une sous-zone de la zone de recherche. Une fois que le processus d'atterrissage a commencé, la méthode ne prend pas en compte les solutions qui ont été faites précédemment, de sorte que l'atterrissage reste concentré sur une zone spécifique de la zone de recherche.

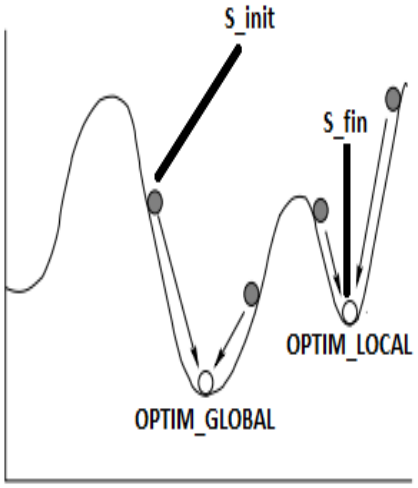


FIGURE 2.2 – L'effet du choix de la solution initiale sur le résultat de la recherche par la Méthode Descente

L'inconvénient de cette méthode est qu'elle s'arrête au premier minimum local. Pour éviter d'être bloqué au premier minimum local : le choix de $N$, peut avoir une grande influence sur l'efficacité de DM. Aussi accepter, sous certaines conditions, de passer d'une solution à une solution adjacente $s' \in N(S)$ telle que $f(s') \geq f(s)$. C'est ce que font les méthodes Recuit Simulé.

### 2.3.2 Méta-heuristique, et voisinages et amélioration locale

Dans les problèmes d'optimisation difficiles, il n'y a pas encore d'approche classique connue et performante qui permette de trouver au moins des solutions optimales. Pour orienter le processus de recherche, il a y des stratégies sont nécessaires, ces stratégies peuvent nous garantir une utilisation correcte de l'espace de recherche [110], comme les problèmes de la recherche opérationnelle, de la bio-informatique, de l'intelligence artificielle, etc.

Pour gérer facilement l'espace de recherche, nous utilisons les méta-heuristiques, qui ont été considérées comme une manière de résoudre des problèmes d'optimisation difficiles. Il existe globalement deux familles de méta-heuristiques : les recherches locales, aussi appelées méthodes à solution unique et les méthodes à population de solutions aussi appelées méthodes évolutionnaires (par exemple les algorithmes génétiques, la recherche dispersée, l'optimisation par des colonies de fourmis, la méthode de la mémoire adaptative... etc). On peut différencier les méta-heuristiques qui font évoluer une seule solution

sur l'espace de recherche à chaque itération et les méta-heuristiques à base de population de solutions. En général, les méta-heuristiques basées sur une solution unique sont plus axées sur l'exploitation de l'espace de recherche, de sorte que vous ne pouvez jamais être sûr d'obtenir l'optimum. Les méta-heuristiques basées sur la population sont plutôt exploratoires et permettent une meilleure diversification de l'espace de recherche [28].

La recherche locale est une méta-heuristique utilisée pour résoudre des problèmes d'optimisation, c'est-à-dire des problèmes où l'on cherche la meilleure solution dans un ensemble de solutions candidates. La recherche locale consiste à se déplacer d'une solution à une autre solution proche dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution considérée comme optimale soit trouvée, ou que le temps imparti soit dépassé.

Les méthodes de recherche locale sont basées sur la notion de voisinage et sur la procédure d'exploitation de ce voisinage. Généralement, les recherches locales sont différenciées sur la base de cette procédure. Pour une recherche de voisinage à très grande échelle, le voisinage est de grande taille et peut-être de taille exponentielle. Les techniques de recherche de voisinage tentent de trouver une nouvelle solution en transformant la solution actuelle, de manière répétée, dans le voisinage de cette dernière, jusqu'à ce qu'une meilleure solution soit trouvée, ou que le critère d'arrêt soit atteint. Le voisinage d'une solution, est l'ensemble de toutes les solutions qui peuvent être obtenues par une simple modification d'une petite partie de la solution candidate, par exemple, la modification d'un seul nœud dans un arbre couvrant génère un arbre différent. Pour cela, il faut définir une notion de voisinage sur l'espace de recherche [132].

Nous présenterons ci-dessous les méthodes les plus utilisées : la recherche locale itérée, le recuit simulé, la recherche tabou et la recherche dans le voisinage à très grande échelle, les algorithmes génétiques et colonie de fourmis, PPA, et les méthodes hybrides.

#### 2.3.2.1 La recherche locale itérée (ILS)

Les méthodes de descente (DM) sont utilisées avec succès pour les problèmes d'optimisation combinatoire nécessitant une solution rapide, mais l'inconvénient de cette méthode est qu'elle s'arrête au premier optimum local trouvé (après un optimum local). Pour éviter la stagnation au niveau de l'optimum local, de nombreuses stratégies sont mises

en place pour continuer les recherches dans d'autres zones du voisinage, cependant, un critère d'arrêt doit être défini, et ce critère peut être le nombre maximum d'itérations. Il peut s'agir du temps d'exécution, voire du nombre d'itérations sans amélioration.

Principe de représentation de l'itération de la recherche locale itérative : Dans un premier temps, afin de générer une meilleure solution, une recherche descendante est appliquée à une solution initiale. Deuxièmement, une nouvelle recherche descendante est appliquée à cette nouvelle solution après l'avoir perturbée. Troisièmement, une comparaison entre la solution résultante et la solution initiale pour voir si elle va la remplacer ou non.

L'algorithme 2.2 présente en pseudo-code une recherche locale itérative classique.

**Algorithme 2.2** Algorithme de la Recherche Locale Itérative Classique

1. **Données** : Une solution initiale ;
2. **Résultat** : Une solution meilleure (ou au moins égale) à $\ll S \gg$ ;
3. $S_*$= résultat d'une recherche locale sur $S$ ;
4. **Répété**
5. $S$‘ = résultat d'une perturbation de $S_*$ ;
6. $S'_*$ = résultat d'une recherche locale sur $S$‘ ;
7. $S_*$ =$S_*$ ou $S'_*$ selon un critère d'acceptation ;
8. Jusqu' á le critère d'arrêt soit vérifié
9. **Retourner** $S_*$

La recherche locale itérée (ILS) : elle est simple, facile à mettre en œuvre, robuste et très efficace. L'idée essentielle de l'ILS est de concentrer la recherche non pas sur l'espace complet des solutions, mais sur un sous-espace plus petit défini par les solutions localement optimales pour un moteur d'optimisation donné. C'est une bonne méthode pour surmonter un arrêt rapide à un optimum local. En appliquant une stratégie de perturbation de la solution courante suivie de la méthode de descente. La décision d'accepter ou de rejeter la nouvelle solution est prise en fonction du critère d'acceptation. Ces étapes se poursuivent jusqu'à ce que le critère d'arrêt soit satisfait. ILS peut souvent devenir un algorithme compétitif, de haut niveau. Tous les processus précédents sont détaillés dans la figure 2.3.

Nous avons utilisé ILS dans notre contribution présentée au chapitre 5, avec une heuristique de réduction des données, et il a donné de bons résultats.

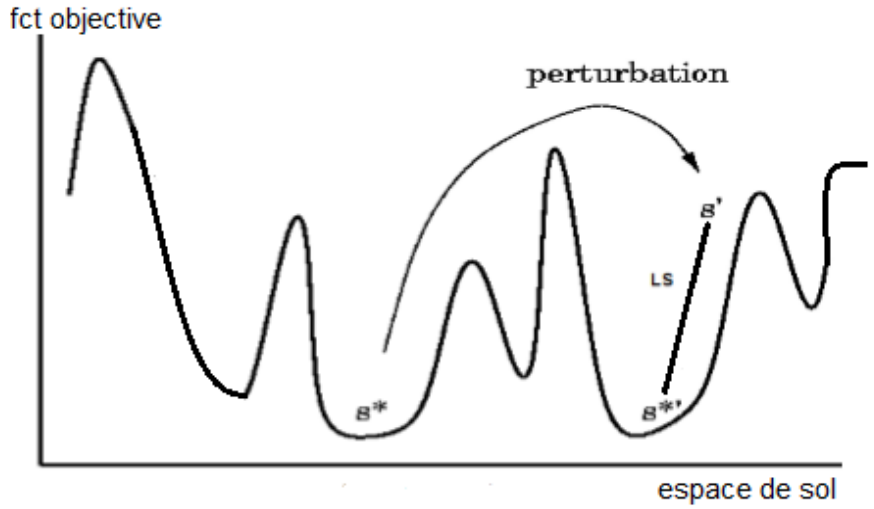


FIGURE 2.3 – La recherche locale itérée

#### 2.3.2.2 La recherche tabou (tabu search, TS)

La recherche tabou est une heuristique de recherche locale. Elle a été définie par Glover [82], comme une amélioration de la méthode de descente. La définition la plus simple est Climbing Hill (également connu sous l'algorithme glouton (greedy algorithm). Où l'on part d'une solution initiale qui est remplacée par une nouvelle solution voisine si elle est meilleure que celle-ci, et cela conduit généralement à trouver une solution locale optimale (voir la figure 2.4).

L'algorithme de Recherche Tabou 2.3 peut être décrit comme suit :

**Algorithme 2.3** L'Algorithme de Recherche Tabou

1. choisir une solution$s \in S$, poser $T \longleftarrow \Phi \quad et \quad s^* \longleftarrow s$ ;
2. **Tant que** aucun critère d'arrêt n'est satisfait **faire**
3. Déterminer une solution $s'$ qui minimise $f$ dans $N_T(S)$ ;
4. **si** $f(s') < f(s)$ **alors**poser $s^* \longleftarrow s'$ ;
5. Poser $s \longleftarrow s'$ et mettre à jour $T$ ;
6. **Fin du tant que**

Un tabou est une structure de la mémoire, il comprend un ensemble de règles et de solutions à court terme qui ont été visitées dans de courtes périodes de temps et leur utilisation est interdite. Lors de l'utilisation de l'algorithme de recherche de tabou, la taille de la liste spécifique de tabou doit être définie avec un certain nombre de caractéristiques, et il est recommandé d'alimenter la liste de tabou avec de nouvelles caractéristiques. Pour surmonter ces restrictions, nous avons besoin d'un nouveau cas qui peut contourner la liste des tabous.

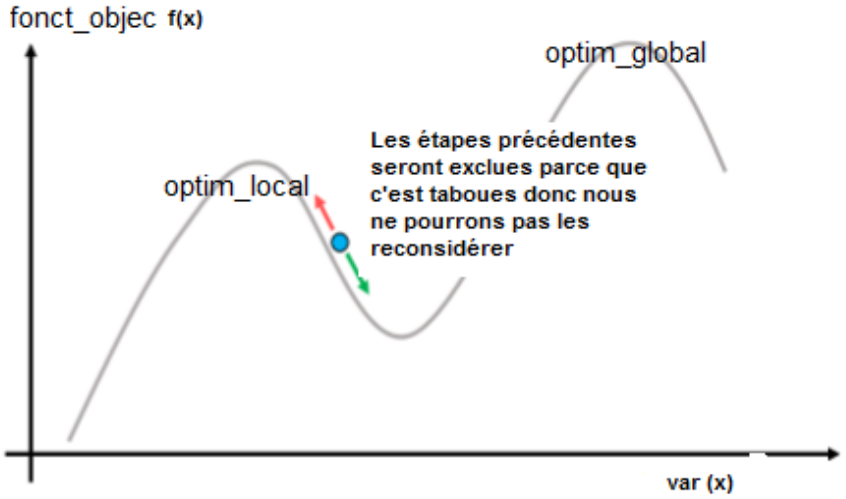


FIGURE 2.4 – La recherche tabou

#### 2.3.2.3 Le recuit simulé (Simulated Annealing, SA)

Le recuit simulé est une méthode d'optimisation apparue en 1982, et son idée principale a été proposée par Metropolis en 1953. Il s'agit de simuler le comportement de la matière dans le processus thermodynamique de recuit des métaux (chauffage et refroidissement progressifs du métal) [27, 115, 184].

Le principe consiste principalement à créer des configurations successives d'une solution initiale $S0$ et d'une température initiale $T0$ qui sera réduite tout au long du processus jusqu'à atteindre la température ou l'équilibre final, soit solution optimale atteint.

Recuit Simulé qui peut être décrit dans 2.4 comme suit :

- 1) La température de départ est choisie $T = T0$, ainsi que la solution initiale $S = S0$.
- 2) Une solution aléatoire est générée au voisinage de la solution actuelle.
- 3) Comparaison des deux solutions selon la norme Metropolis.
- 4) Répétez les étapes 2 et 3 jusqu'à ce que l'équilibre statistique soit atteint.
- 5) Réduisez la température et répétez jusqu'à ce que le système soit gelé.

Le recuit simulé fournit généralement une solution de bonne qualité par rapport aux algorithmes de recherche traditionnels. Mais le principal inconvénient du recuit simulé est qu'une fois que l'algorithme est maintenu à basse température dans le minimum local, il devient impossible de le supprimer. Malheureusement, il est impossible de savoir

**Algorithme 2.4** Algorithme de la recherche par recuit simulé

1. choisir une solution$s \in S$, et poser une température initiale $T$ ;
2. **Tant que** aucun critère d'arrêt n'est satisfait **faire**
3. Choisir aléatoirement une solution $s' \in N(s)$ ;
4. Générer un nombre réel aléatoire $r \in [0, 1]$ ;
5. **Si** $r < p(T, s, s')$ alors poser $s \longleftarrow s'$ ;
6. mettre à jour $T$ ;
7. **Fin du tant que**

si une solution existante est optimale et il est très difficile de déterminer la température initiale (la qualité de la recherche n'est pas satisfaisante en cas de température très basse), d'autre part, si la température est trop élevée, le temps de calcul sera long, donc le réglage des paramètres recommandés est difficile à déterminer et se fait souvent de manière expérimentale, donc un critère d'arrêt est nécessaire si le réglage "optimal" n'est pas trouvé.

Dans la figure 2.5, nous présentons un chemin de recuit simulé dont le but est de trouver des solutions globales minimales. Il ne suffit pas d'utiliser un simple algorithme de Hill Climbing car il existe de nombreux maxima locaux.

Bien que le recuit simulé ait été initialement présenté pour des programmes discrets, il peut être utilisé pour la résolution de programmes continus en réduisant l'espace de recherche.

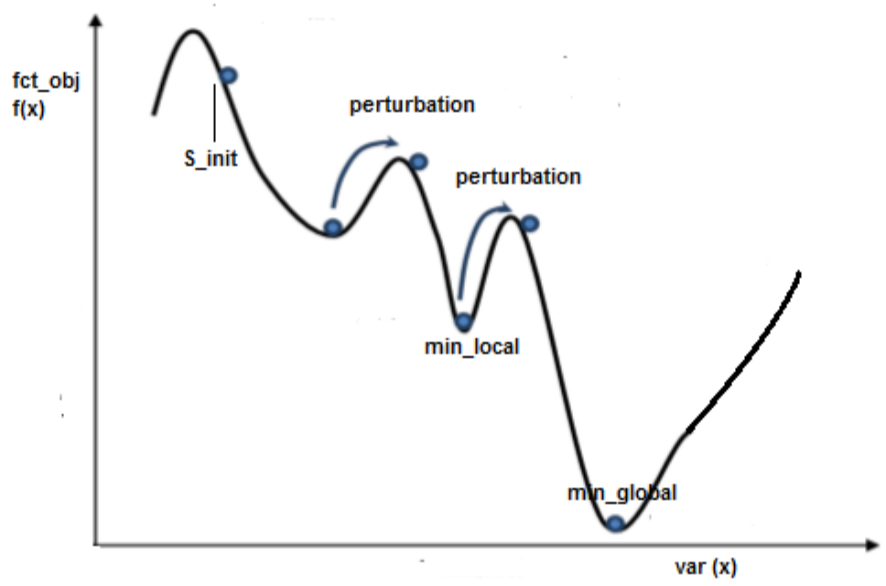


FIGURE 2.5 – Le recuit simulé

#### 2.3.2.4 La recherche dans le voisinage à très grande échelle (very large scale neighborhood, VLSN)

Un algorithme d'optimisation est un algorithme heuristique qui part généralement d'une solution exploitable et essaie à plusieurs reprises d'obtenir une meilleure solution. Les algorithmes de recherche de voisinage sont une grande classe d'algorithmes d'optimisation dans lesquels une solution d'optimisation est trouvée à chaque itération par le processus de recherche du "voisinage" de la solution courante. Le problème critique dans la conception d'un algorithme de recherche de voisinage est de savoir comment choisir une structure de voisinage (comment définir un voisinage ?). En règle générale, plus le voisinage est grand, meilleure est la qualité des solutions localement optimales et meilleure est la précision de la solution finale obtenue. En même temps, plus le voisinage est grand, plus la recherche de voisinage sera longue à chaque itération. Pour cette raison, le plus grand voisinage ne produit pas nécessairement une extension plus efficace, à moins que le plus grand voisinage puisse être trouvé d'une manière très bonne et efficace [4].

L'algorithme de La recherche dans le voisinage à très grande échelle a été implémenté dans plusieurs problèmes d'optimisation combinatoire très importants, tel que le problème de ramassage et livraison [127], le problème de la planification de chargement de fret aérien [126], le problème d'arbre couvrant de poids minimal [3, 5], le problème général d'affectation [80, 227], le problème de regroupement [63], le problème de tournées de véhicules [119, 178], le problème de voyageur de commerce [55, 69], le problème de planification des satellites [129], le problème de planification des machines parallèles[2], le problème d'emploi du temps[136], le problème d'affectation quadratique[7].

Lors du développement de techniques de recherche par voisinage, le principal défi consiste à définir les voisins. Là où la faiblesse de ces méthodes de recherche (de blocage) se situe au niveau local optimal, mais en général, on observe que les grands voisinages ont tendance à trouver un avantage local de haute qualité par rapport aux autres problèmes. Son coût est très élevé par rapport à ses voisins plus petits et il augmente exponentiellement avec la taille du problème. Ainsi, la recherche dans un grand voisinage n'est pas pratique si la recherche n'est pas dirigée par une méthode de recherche efficace. Par conséquent, un algorithme de recherche rapide est essentiel pour une utilisation efficace des grands voisinages.

Les algorithmes de recherche pour les très grands voisinages sont divisés en trois classes principales, comme le montre l'illustration ci-dessous [4] :

— La 1$^{er}$ classe comprend les méthodes à profondeur variable (variable-depth methods) qui appliquent une heuristique à une recherche partielle dans un voisinage très vaste et en croissance.

— La 2$^{e}$ classe contient des algorithmes d'optimisation basés sur le flux réseau (network flow based improvement algorithms) pour trouver des voisinages et identifier des voisins améliorés à l'aide de techniques de flux réseau.

— La 3$^{es}$ classe comprend des voisinages basés sur des restrictions de problèmes NP-difficiles qui peuvent être résolus en temps polynomial.

les études suivantes : Ahuja *et al.* [4, 6], ils fournissent une présentation approfondie des algorithmes de ces trois classes liés par les méthodes de voisinage à très grande échelle.

Lorsque les méthodes de recherche VLSN sont utilisées pour la plupart des problèmes, les résultats de calcul sont presque toujours bons et meilleurs, ce qui rend le développement de tels algorithmes hautement souhaitable. La conception réussie de l'algorithme de recherche VLSN repose sur la sélection de la fonction de voisinage correcte et sur le développement d'une méthode heuristique efficace pour rechercher le voisinage et optimiser les solutions.

#### 2.3.2.5 Les algorithmes génétiques

Un algorithme génétique est un algorithme d'optimisation qui se base sur des aspects dérivés de la génétique et de l'évolution naturelle, tels que le transit, la mutation et la sélection, la représentation des chromosomes et les opérateurs génétiques. L'algorithme génétique a une origine qui commence en 1962 lorsque John Holland [102], a effectué des travaux sur les systèmes adaptatifs. Il trouve la meilleure fonction spécifique dans l'espace de données. Pour fonctionner, il doit y avoir cinq éléments [54, 102, 184] : (1) Un principe de codage de l'élément de population, (2) Un mécanisme de génération de la population initiale, (3) Une fonction à optimiser, (4) Des opérateurs permettant de diversifier la population au cours des générations, et (5) Des paramètres de dimensionnement.

Dans ce travail Debbat and Bendimerad [54], l'objectif qu'ils soumettent est une étude et une évaluation de l'adéquation de deux algorithmes d'optimisation (recuit simulé et

algorithme génétique) dans le cadre de l'optimisation d'un réseau d'antennes intelligent (adaptatif).

Le fonctionnement de l'algorithme génétique est le suivant :

— Étape 1 : l'algorithme génétique commence par une génération d'une population initiale de taille $P_{size}$.

— Étape 2 : évaluation des $P_{size}$ individus.

— Étape 3 : sélection pour la reproduction.

— Étape 4 : croisement.

— Étape 5 : mutation.

— Étape 6 : évaluation des individus enfants.

— Étape 7 : insertion.
répéter l'étape 3 à 7 jusqu'à ce que les critères d'arrêt soient vérifiés.

#### 2.3.2.6 L'algorithme de colonie de fourmis (ACO)

Au début des années 1990, la conception et la structure de l'algorithme de colonie de fourmis (aussi appele ACO pour Ant Colony Optimization) inspiré de la nature ont été conçues par Dorigo and Blum [62]. Il est basé sur le rythme particulier des fourmis qui cherchent de la nourriture, explorent leur environnement et laissent de l'acide formique, pour finalement développer souvent les chemins les plus courts entre une fourmilière et une source de nourriture intéressante.

Chaque fourmi attire l'attention de ses collègues en triant les phéromones qu'elles laissent derrière elles. Les fourmis sélectionnent très probablement les chemins qui présentent les plus fortes densités de phéromones, formant ainsi ces "autoroutes" à fourmis.

Comme il est défini par Dorigo and Blum [62], ACO fonctionne comme suit :

— Étape 1 : chaque fourmi crée une solution globale en sélectionnant des tours selon une loi de passage d'état aléatoire.

— Étape 2 : les fourmis privilégient les tournées qui sont connectées par un parcours court avec une grande quantité de phéromone sur le parcours entre les deux tournées.

— Étape 3 : une fois les fourmis finies et chacune obtenant une solution, une nouvelle règle de mise à jour des phéromones est effectuée. Une fraction des phéromones se volatilise sur tous les parcours (les parcours qui ne sont pas actualisés deviennent moins souhaitables), et chaque fourmi dépose une dose de phéromone sur les parcours qui appartiennent à ses tournées en proportion de la qualité de chaque tournée.

— Étape 4 : le déroulement est ensuite répété jusqu'à ce que les conditions d'arrêt soient satisfaites.

Ainsi, le comportement des fourmis est déterminé par deux règles principales : la sélection des parcours et la mise à jour des phéromones.

#### 2.3.2.7 L'algorithme de propagation des plantes (PPA)

Certains chercheurs ont fait la preuve que les plantes présentent des comportements intelligents. Bien qu'il existe treize algorithmes méta-heuristiques de recherche et d'optimisation basés sur l'intelligence des plantes, ils ne sont pas bien connus des chercheurs du domaine connexe. Toutefois, ils semblent être très demandés pour résoudre des problèmes importants et difficiles en raison de leur grande capacité et efficacité [8, 29].

Une nouvelle famille d'algorithmes d'optimisation combinatoire a été décrite, basée sur la propagation des plantes en particulier des fraises, avec la mise en avant d'une implémentation spécifique simulant le comportement des fraises. Il s'agit d'une méta-heuristique basée sur la population.

L'algorithme de propagation des plantes (PPA) est une méta-heuristique, qui a été inspirée par la nature et appelée Plant Propagation Algorithm, a été développée par Salhi and Fraga [183], et c'est l'une des méta-heuristiques les plus récentes qui ont été identifiées avec un fort succès dans la solution de problèmes d'optimisation combinatoire difficiles. En effet, l'APP généralise la recherche locale itérée [131].

Dernièrement, plusieurs études ont abordé l'algorithme PPA dans un grand nombre d'articles de recherche, nous mentionnons les suivants : Planification de parcours [232], Voyageur de commerce [189], Planification de l'examen [39], L'optimisation de l'ingénierie [208], Répartition du chargement [146]. En outre, d'autres recherches se concentrent sur

l'ajout de diverses modifications et améliorations à PPA, telles que : Sulaiman and Salhi [206, 207], Sulaiman *et al.* [209]. Il existe également des études récentes qui contient plus de détails sur la PPA tels que : Borzog-Haddad *et al.* [26], Geleijn *et al.* [76], Paauw and Van Den Berg [158].

Contrairement aux méthodes classiques, les algorithmes inspirés par la nature se sont prouvés très efficaces dans les problèmes d'optimisation difficiles. En revanche, les algorithmes exploratoires et les procédures heuristiques inspirées de la nature présentent les inconvénients suivants : ils ont une théorie assez limitée pour les prendre en charge, et sont aussi principalement stochastiques, c'est-à-dire basés sur de nombreux paramètres souvent aléatoires, cela signifie que les résultats changent d'une série à l'autre et qu'il peut être difficile d'obtenir des résultats identiques de différents chercheurs [183].

Le principe de base est mentionné en pseudo-code, voir 2.5.

PPA a commencé avec un ensemble de plantes réparties aléatoirement dans l'espace de recherche (première ligne de l'algorithme 2.5). Le nombre $\%good$ (ligne 4) est le pourcentage de plantes considérées comme étant dans des environnements appropriés. Un tri dans lignes 2 et 7 est nécessaire pour distinguer les plantes $S$ dans les bons endroits et les plantes $N - S$ aux mauvais endroits. Dans la ligne 8 de l'algorithme 2.5, seul un pourcentage des plantes générées et de leurs parents passables. Cette sélection est essentielle pour maintenir la population à une taille raisonnable.

Comme nous savons que pour toute méta-heuristique, il faut attribuer un ensemble de paramètres, PPA nécessite également l'attribution d'un ensemble de paramètres et la prise de décisions générales : taille de la population $N$, pourcentage de plantes considérées comme étant en bon état et nombre $r$ de petits stolons courts qui ont envoyé de bons parents et nombre $R$ de stolons longs qui ont envoyé des parents aux mauvais endroits $r > R$. D'autre part, il y a des paramètres qui exigent des décisions spécifiques d'un problème : "la courte ou longue distance" parcourue par le stolon, le nombre de plantes à rejeter (Ligne 8) selon le critère d'acceptation.

Pour toute méthode de n'importe quelle méta-heuristique, il y a deux composantes de base, à savoir la diversification et l'intensification (elles sont également d'une importance fondamentale dans l'APP), et il faut atteindre l'équilibre entre elles, car elles ont une

**Algorithme 2.5** Pseudo-code de base pour le PPA

1. Générer la population initiale $p = [p_1, ....., p_N]$ ;
2. Ordonner $p$ par ordre croissant en fonction de la fonction objectif ;
3. **Répété**
4. $S = \%bon * N$ ;
5. **Pour** $i = 1, ..., S$ la plante $p_i$ génère $r$ stolons courts
6. **Pour** $i = S + 1, ..., N$ la plante $p_i$ génère $R$ stolons courts
7. Ordonner la nouvelle population (plus large) ;
8. Sélectionner $N$ plantes de la nouvelle population selon un critère d'acceptation ;
9. **Jusqu'au critère d'arrêt.**
10. **Retourner** $p_1$**.**

action contraire. La phase de diversification offre la possibilité à la recherche d'échapper à la zone d'attraction des optima locaux. Alors que la phase d'intensification se concentre sur la recherche au voisinage des optima locaux. L'intensification et la diversification sont d'une importance fondamentale dans l'APP, sachant que l'intensification est réalisée en envoyant un petit stolon des usines mères vers les endroits les plus pratiques et efficaces. Quant à la mise en œuvre de la diversification, elle se fait par l'envoi d'un long stolon des plantes mères vers des lieux inadaptés, c'est-à-dire des lieux arides. La stratégie de survie du plant de fraisier dans son environnement peut donc être considérée comme celle de la recherche de points dans l'espace des solutions qui permet finalement d'obtenir de meilleures valeurs de forme.

#### 2.3.2.8 Méthodes hybrides

La méta-heuristique est une nouvelle approche avec de solides techniques d'approximation globale, qui sont applicables à une grande classe de problèmes d'optimisation combinatoire.

La méta-heuristique 2.6 est spécifiquement conçue pour trouver la solution globale idéale (optima_global) à un problème d'amélioration difficile, en tenant compte des pauses, elle évite de tomber dans le piège de l'idéalisme local (optima_locaux).

Le diagramme 2.7 présente les différentes méthodes : Les méta-heuristiques classent un des cas de plusieurs façons et distinguent celles qui travaillent avec une population de solutions et celles qui ne manipulent qu'une solution à la fois. Nous nous intéressons aux algorithmes des méthodes hybrides.

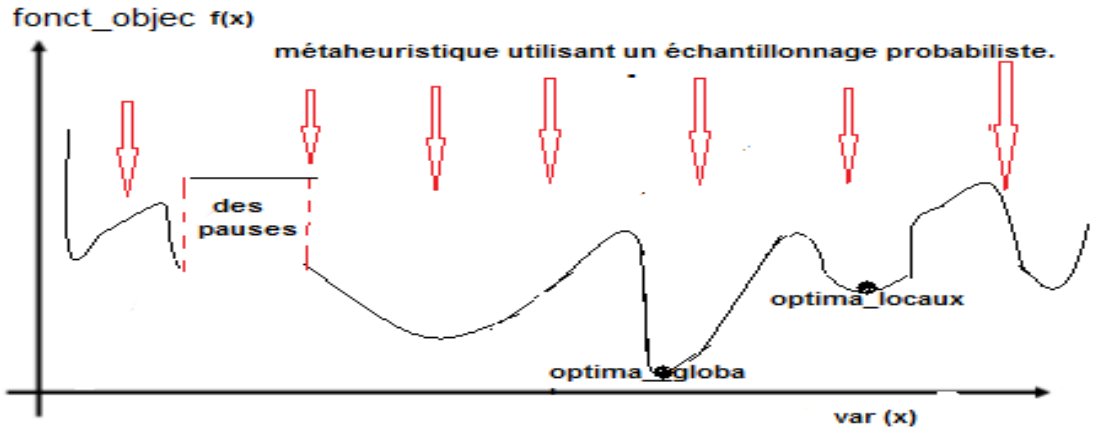


FIGURE 2.6 – Les méta-heuristiques.

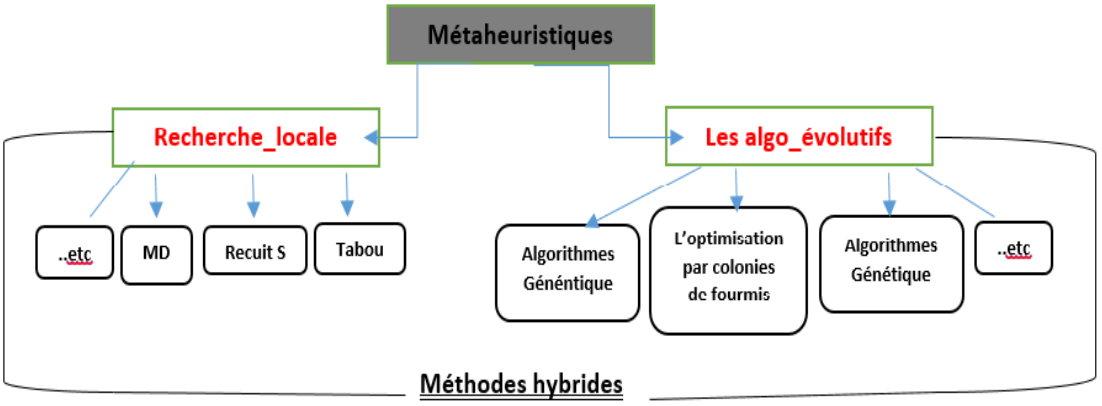


FIGURE 2.7 – Les méta-heuristiques et les méthodes hybrides .

Ces dernières années, la recherche sur les méta-heuristiques a montré ce que l'on appelle l'hybridation, définie comme une tendance à tirer parti des avantages cumulatifs des propriétés méta-heuristiques pour commencer à résoudre le problème d'optimisation combinatoire difficile.

L'une des techniques d'hybridation les plus populaires est l'utilisation de méta-heuristiques de type "une solution" en conjonction avec l'une des méta-heuristiques de type "population de solutions". Par exemple, la plupart des implémentations réussies des algorithmes de colonies de fourmis peuvent être complétées par une étape de recherche locale, car c'est ce qui lui manquait à l'origine. Les individus et les fourmis tentent d'identifier des zones prometteuses dans l'espace des solutions, qui sont ensuite explorées plus en détail par des méthodes de recherche locale (par exemple : le plus proche voisin ou par des heuristiques classiques, etc). Ces deux types de recherche (générale et locale) peuvent également être effectués de manière asynchrone sur différents processeurs.

Il existe également une autre méthode d'hybridation, cette méthode consiste à exécuter plusieurs fois en parallèle la même méta-heuristique avec des paramètres différents.

Enfin, une autre forme d'hybridation combine des méthodes exactes avec des méta-heuristiques. Une méthode exacte peut conduire à une méthode efficace qui détermine le meilleur voisin d'une solution et cette solution courante peut être efficace plutôt que de choisir la meilleure solution parmi un petit échantillon de voisins. Dans le domaine de la programmation par contraintes, certaines techniques de régression non déterministes ont été associées à des méta-heuristiques.

Les algorithmes mémétiques MA (ont été introduits par Dawkins [52], et formalisés par Moscato [141]). Ils ont une hybridation entre les algorithmes génétiques et les algorithmes de recherche locale.

Nous allons maintenant décrire l'algorithme MA 2.6 :

— 1 : L'initialisation de *pop* un premier groupe d'individus (cette initialisation peut être gloutonne ou aléatoire).

— 2 : L'pplication de l'opérateur de sélection le choix de ces individus

— 3 : L'application de l'opérateur de croisement. cet opérateur est utilisé pour créer un nouvel individu à partir de deux individus parents.

— 4 : L'application de l'opérateur de recherche locale aux nouveaux individus créés (enfants).

— 5 : L'application de l'opérateur de mutation permet d'introduire de nouveaux gènes dans la population. (Pour quitter l'optimum local).

— 6 : Mise à jour remplacer $P$ le plus mauvais parent par $C$.
Remarque : le critère d'arrêt est différent d'un problème à un autre.

D'autre part, les algorithmes de recherche VLSN peuvent être développés pour être robustes et efficaces en les hybridant avec d'autres méthodes heuristiques et méta-heuristiques afin d'arriver à des solutions nouvelles et améliorées, telles que la recherche locale itérée [212], la recherche dispersée [4, 6, 179], la recherche tabou [101, 109], et la recherche à voisinage variable [66].

Ces récentes années, l'hybridation est devenue une véritable tendance dans la résolution des problèmes d'optimisation combinatoire. Les méthodes exactes donnent généralement

**Algorithme 2.6** Pseudo-code de MA hybride

1. Générer la population initiale $POP$ de solution avec taille $=n$ ;
2. Améliorer chaque $s$ de $POP$ : $s \leftarrow RL(s)$ ;
3. **Répété**
4. Avec la technique de sélection, sélectionner deux solutions $x$ et $x'$ ;
5. Croiser les deux parents $x$ et $x' \rightarrow$ enfants $C1, C2$ ;
6. **Pour chaque** enfant $C$ **Faire** ;
7. Améliorer : $C \leftarrow RL(C)$ ;
8. Muter $C$ ;
9. Remplacer une solution $P$ de $POP$ par $C$ ;
10. **Fin pour**
11. **Jusqu'au critère d'arrêt.**

les meilleurs résultats en termes de qualité de solution, mais au coût de la disponibilité. Face à un problème d'optimisation difficile et de grande taille, il est donc nécessaire de se tourner vers des heuristiques (pour un problème spécifique) ou des méta-heuristiques (plus généralement) pour gagner du temps d'implémentation et parfois de combiner les deux approches (exacte et heuristique) ou deux méta-heuristiques bénéficiant des deux.

## 2.4 Conclusion

Comme tout problème d'optimisation combinatoire, les méthodes de résolution sont classées en deux catégories : Exactes et approchées. Parmi les méthodes exactes, nous présentons dans ce chapitre, la programmation linéaire en nombres entier et dynamiques, la relaxation lagrangienne, le Branch and Bound (B&B), et l'algorithme glouton. Ces approches sont des méthodes de recherche classiques pour des problèmes de taille raisonnable. Cependant, si le problème est très compliqué pour avoir une solution exacte, nous utilisons des méthodes approchées pour trouver des solutions acceptables et de bonne qualité dans un temps d'exécution raisonnable. Parmi ces méthodes, nous introduisons les heuristiques, les métaheuristiques.

Parmi les méthodes les plus prometteuses et les plus efficaces, nous nous concentrerons dans la suite du chapitre sur les métaheuristiques qui sont classées en deux catégories, à savoir :

1. Les métaheuristiques basées sur la recherche locale, où la recherche commence par une solution initiale, puis se déplace itérativement vers une solution voisine. Parmi

ces méthodes, nous avons présenté : le recuit simulé, la recherche par table, ILS, etc.

2. Les métaheuristiques basées sur la population, où la recherche commence avec un ensemble de solutions initiales pour explorer différents lots de l'espace de recherche, puis elles sont recombinées itérativement pour générer une nouvelle solution. Parmi ces méthodes nous avons cité : les algorithmes génétiques, la colonie de cercles, etc.

# Chapitre 3

# Gestion de mémoire dans les systèmes embarqués – Un état de l'art

## 3.1 Introduction

Les systèmes embarqués sont l'une des technologies les plus développées et souhaitées dans tous les domaines jour après jour. De nombreuses définitions sont mentionnées dans la littérature scientifique, qui s'intéresse aux systèmes embarqués publiés dans Noergaard [151]. Un système embarqué est un système informatique fonctionnel ; il se distingue des autres systèmes informatiques, par exemple les ordinateurs personnels ou les superordinateurs. Un système embarqué est destiné à effectuer une tâche spécifique au sein d'un système mécanique ou électrique plus étendu.

La plupart des systèmes embarqués sont principalement conçus pour exécuter des tâches spécifiques, fonctionner dans une routine précise et effectuer certaines opérations prédéfinies.

Cependant, de nos jours, nous considérons de nombreux appareils comme des téléphones mobiles destinés à remplir différentes fonctions. En outre, les derniers téléviseurs contiennent également de nombreuses applications interactives qui exécutent des tâches différentes sans rapport avec la fonction principale du téléviseur [151].

En général, les composants de base d'un système embarqué : processeurs, bus, I/O, mémoire et logiciel. Les systèmes embarqués présentent quelques inconvénients : certaines modifications peuvent entraîner des problèmes et la gestion des crises est très difficile ; de plus, il est difficile de transférer des données d'un système à l'autre.

Le principal avantage des systèmes embarqués est qu'ils sont conçus pour gérer certaines tâches simples, mais il faut noter que les étapes de gestion ou d'exécution de ces tâches peuvent être aussi complexes que n'importe quel programme informatique. Parmi les caractéristiques des systèmes embarqués modernes, mentionné dans Artes *et al.* [11], où les systèmes embarqués ont des caractéristiques différentes et explicites par rapport aux systèmes à usage général. Premièrement, les systèmes embarqués combinent à la fois le logiciel et le matériel pour exécuter un ensemble fixe et particulier d'applications. Ces applications sont très différentes dans leurs caractéristiques. Deuxièmement, contrairement aux systèmes à usage général, les systèmes embarqués sont caractérisés par des ressources limitées et une faible consommation d'énergie, ce qui limite leur cycle de fonctionnement. Troisièmement, afin d'être prévisibles et dépendants, les systèmes embarqués peuvent se permettre des capacités de calcul élevées lorsqu'ils répondent aux

contraintes de contention temporelle diverses et serrées qui sont imposées par l'application en cours d'exécution.

Comme on le sait, les systèmes embarqués sont principalement conçus dans le cadre de budgets de consommation d'énergie limités, afin d'éviter les émissions de chaleur de la batterie et de réduire sa taille. Comme les systèmes de mémoire consomment une grande quantité d'énergie pour stocker les données et les transmettre, il est crucial dans la phase de conception du système de mémoire d'équilibrer la consommation d'énergie et les performances [23], et c'est ce à quoi s'intéresse notre étude actuelle. La mémoire embarquée commence à être l'une des parties les plus essentielles des processeurs et des systèmes sur puce, car elle a une implication positive sur la performance ; mais la mémoire embarquée peut avoir un impact négatif sur la surface, la puissance et le temps. Pour réduire l'écart entre les fréquences des processeurs et le temps d'accès aux DRAM, il est fortement recommandé d'utiliser de plus en plus de mémoires sur puce avec de la mémoire embarquée.

Pour atteindre l'objectif global de performance du système, nous devrions accorder plus d'attention à la conception et à la hiérarchie des sous-systèmes de mémoire afin de fournir à l'unité d'exécution les données et les instructions nécessaires aussi rapidement que possible. Les processeurs peuvent parfaitement fonctionner, et les performances à 100% ne sont garanties que dans un seul cas, celui où la taille de la mémoire est si énorme et où l'accès à la mémoire est proche de zéro seconde, ce qui est l'objectif de la conception et de la hiérarchie des sous-systèmes [139].

L'allocation de mémoire est l'un des plus grands défis pour les concepteurs et les développeurs de systèmes embarqués. Les mémoires importantes peuvent fournir une utilisation élevée d'énergie, mais une allocation de mémoire soigneuse des données peut améliorer la vitesse d'accomplissement des tâches les cœurs de processeur essayent d'atteindre la meilleure performance, mais l'écart entre les fréquences de processeur et l'accès à la mémoire forme le goulot d'étranglement pour obtenir la meilleure performance. La réduction de l'utilisation de l'énergie aurait un impact positif sur la performance [201]. Un état de l'art des techniques d'optimisation pour la gestion de la mémoire et l'affectation des données est présenté les sections ci-dessous. De nombreux chercheurs ont proposé des solutions de gestion de la mémoire pour réduire la consommation d'énergie et augmenter les performances globales du système.

Comme cela a été présenté ici, le système de mémoire est la principale source de performance et de consommation d'énergie. Par conséquent, les systèmes embarqués sont connus pour leur faible consommation d'énergie, il devient donc crucial de rechercher des méthodes d'optimisation de la mémoire pour ces systèmes. Il existe de nombreuses méthodes pour diminuer et réduire la consommation d'énergie et le temps d'exécution des systèmes embarqués que nous allons aborder ici.

Ce chapitre 3 présente le contexte des techniques d'optimisation de la mémoire pour les systèmes embarqués. Tout d'abord, nous commencerons par expliquer et décrire l'optimisation matérielle 3.3, qui aborde certaines manières de promouvoir la conception de l'architecture de la mémoire embarquée. Deuxièmement, nous expliquerons et décrirons également l'optimisation logicielle 3.4, plus précisément la promotion logicielle, en d'autres termes, le développement du code source de l'application séparément de l'architecture. Troisièmement, nous expliquerons et décrirons l'optimisation matérielle/logicielle 3.5, le concepteur est amené à corréler les données de l'application en mémoire à une structure établie.

## 3.2 Techniques d'optimisation de mémoire dans les systèmes embarqués

L'objet principal de notre étude concerne la mémoire et la manière d'améliorer ses performances. La gestion de la mémoire a une grande influence sur le coût global : consommation d'énergie, espace et performance. Les concepteurs de systèmes embarqués doivent réduire les besoins en mémoire ainsi que la consommation d'énergie. L'objectif principal est de réduire le coût global du système. Il existe de nombreuses techniques dans la littérature pour améliorer la mémoire dans les systèmes embarqués. Ces techniques se concentrent sur des améliorations matérielles et logicielles ou les deux, que nous allons présenter ici.

Comme mentionné précédemment, l'organisation de la mémoire s'est avérée être un moyen efficace d'améliorer la consommation d'énergie et la disponibilité. La consommation d'énergie est devenue l'un des bottlenecks des systèmes de mémoire [11].

Ensuite, nous discuterons et réviserons les techniques qui réduisent la consommation d'énergie des systèmes embarqués. Nous les avons classées en plusieurs types de méthodes.

Ces différentes méthodes peuvent être divisées en plusieurs catégories. Selon Panda *et al.* [160] et Benini *et al.* [23], ils ont proposé des idées sur les techniques d'optimisation de la mémoire. Les approches de conception de mémoire à faible consommation existant dans la littérature [160] et [23]. Ils ont mentionné les points suivants :

* Optimisation du matériel : Cette approche aborde de nombreuses routines pour développer la mémoire au niveau structurel et améliorer la consommation énergétique des structures de mémoire.
* Optimisation du logiciel : Cette approche se concentre sur l'optimisation de l'efficience du code et la diminution de la complexité, ainsi que sur la diminution des temps d'exécution critiques dangereux.
* Liaison de données (co-conception matériel/logiciel) : Cette approche permet de bénéficier de l'avantage le plus significatif des structures de données d'une application distribuée à une architecture à mémoire distribuée.

Dans le travail présenté par Benini *et al.* [23], il s'est concentré uniquement sur les techniques centrées sur le matériel. Cependant, un inconvénient a été constaté. L'étude n'était pas très approfondie en termes d'organisation des instructions mémoire. Dans le travail présenté par Panda *et al.* [160], est un résumé des techniques d'optimisation, qui visait réduire l'énergie comme objectif principal ; un inconvénient a été trouvé dans ce travail aussi, qui comprend uniquement l'optimisation liée aux Logiciels, et il n'a pas discuté la signification de nombreuses plates-formes parallèles, y compris les données et le parallélisme (dynamique) au niveau des tests.

Dans le tableau 3.1, nous essayons de montrer les méthodes de calcul intelligentes et les techniques de solution actuellement appliquées à l'optimisation de la mémoire dans les systèmes embarqués [23, 160].

TABLE 3.1 – Principaux algorithmes et techniques de pointe utilisés dans les problèmes d'allocation de mémoire dans les systèmes embarqués

| | | Techniques de solution et méthodes intelligentes | | | |
|---|---|---|---|---|---|
| Optimisation des techniques dans les systèmes embarqués | Liaison de données (HW-SW) | Approches heuristiques | Métaheuristiques | | |
| | | | Hyper Heuristique | | |
| | | Nouvelles Méthodes | Approche hybride | | |
| | | | Approches en logique floue | | |
| | Optimisation du matériel | au niveau du circuit | Mise en œuvre et conception Modules SRAM et DRAM | Réduction de la consommation active | |
| | | | | Réduction de la consommation d'énergie | |
| | | au Niveau d'architectures | Implémenter une mémoire à travers une architecture plate | Mémoires partitionnées | Utilisation de tampons en même temps que les caches |
| | | | | | Partitionnement des fichiers de registre aux mémoires hors puce |
| | | | | | partitionnement physique/logique |
| | | | | Utilisation de la technique pour la lecture plusieurs mots en un seul accès | Insertion de mémoires ad hoc entre les niveaux hiérarchiques existants |
| | | | | | Boucle Caches pour stocker l'instruction la plus fréquemment exécutée |

suite de la page précédente

**Table 3.1 – suite de la page précédente**

| | | | | | |
|---|---|---|---|---|---|
| | | Techniques de solution et méthodes intelligentes | | | |
| | | | | | Remplacer les caches par Scratchpad |
| | | | | | Application Mémoire spécifique ASM |
| | | | | Réduiser le taux d'échec du cache grâce à une conception de cache appropriée (cache associative de Colum/Skewed) | |
| | | | | Optimisation de la bande passante (Code Techniques de compression) | |
| | | Autres méthodes | Affectation des registres par Coloration de graphe | | |
| | | | Attribution de variables scalaires à des mémoires à port unique et à ports multiples | | |
| | | | Modélisation des accès mémoire en HLS | | |
| | | | Commande et réduction de la bande passante | | |
| | | | Packing de mémoire et affectation tableau-mémoire | | |
| | | | Exploration du temps d'accès à la mémoire en utilisant les diagrammes de Pareto | | |
| | | | Réduction des transitions de bus (encodage, organisation des données) | | |
| | | | Réduire la taille de la mémoire requise | | |
| | | | Planification de l'accès au cache et disposition des données | | |

suite de la page précédente

**Table 3.1 – suite de la page précédente**

| | | | |
|---|---|---|---|
| | Techniques de solution et méthodes intelligentes | | |
| | | Séparation des caches spatiaux et temporels | |
| | | Modélisation de DRAM pour HLS et optimisation | |
| | | Synchrone DRAM / Optimisation Banking | |
| | | Gestion de la mémoire Compilation | |
| | | hardware de génération d'adresses | |
| | Optimisation des logiciels | Techniques de réécriture de code pour améliorer la réutilisation des données | |
| | | Techniques de réécriture de code pour la localité et la régularité d'accès | |
| | | Parallélisation des tâches et des données | |
| | | Allocation dynamique de mémoire | |
| | | Les techniques d'estimation | |

## 3.3 Gestion de mémoire orientée-matériel (optimisation matérielle)

La hiérarchie de la mémoire peut être une mémoire cache ou une mémoire scratch-pad (SPM), la mémoire principale. Grâce à de nombreuses études, la caractéristique de cache est la mémoire la plus rapide, et elle consomme moins d'énergie que la mémoire principale. Contrairement à la Scratch-Pad (SPM) qui présente des singularités avec la grande vitesse de la mémoire interne. Cependant, la mémoire principale DRAM occupe une grande surface, et elle consomme tellement d'énergie, et en plus elle perd beaucoup de temps pendant l'accès à la mémoire. Dans cette section, nous essaierons de voir comment nous pouvons optimiser la conception de l'architecture de la mémoire (mise à niveau du matériel).

Par exemple, l'étude présentée par Panda *et al.* [160] a exploré en profondeur l'optimisation de la mémoire par la modélisation, la personnalisation et l'optimisation des stratégies conçues pour les architectures de mémoire cibles à différents niveaux de granularité, en commençant par les registres et les fichiers de registre jusqu'à la SRAM, le cache et la DRAM.

Nous allons maintenant examiner une taxonomie des techniques d'amélioration de la mémoire dans les approches centrées sur le matériel, classées comme suit [23] :

— Techniques d'exploration : La première catégorie de manières est nécessairement exploratoire dans le sens où elle permet une recherche complète ou semi-complète. Les efforts les plus extraordinaires dans ce domaine supposent une hiérarchie de mémoire avec un ou plusieurs niveaux de cache et une mémoire hors de la puce dans de nombreux états. En respectant un nombre limité de tailles de cache et d'options d'organisation du cache. L'organisation de la mémoire la plus excellente est obtenue en affectant la charge de travail de toutes les structures possibles. Les diverses participations à la recherche diffèrent par le nombre de niveaux de régime liés à la recherche ou par le nombre de dimensions possibles dans le domaine de la conception des formulaires.

— Partitionnement de la mémoire : Il s'agit d'une solution standard de réduction de l'énergie axée sur les performances, car elle réduit la latence en entrant dans

de petits morceaux de mémoire et l'énergie ne peut être diminuée que pour certains modèles d'entrée particuliers. Les techniques basées sur le partitionnement proposées dans la littérature varient de plusieurs points de vue. Tout d'abord, le niveau hiérarchique visé par le partitionnement (des fichiers de registre aux mémoires hors puce). Ensuite, le "type" de partitionnement : le partitionnement physique répartit sûrement l'espace d'adressage sur différents blocs de mémoire qui ne se chevauchent pas ; et le partitionnement logique permet une certaine redondance dans les différents blocs de la partition, avec la possibilité d'avoir des adresses qui sont répliquées plusieurs fois dans le même niveau d'une hiérarchie. De nombreux contributeurs ont analysé le partitionnement physique des mémoires embarquées.

— Extension de la hiérarchie de la mémoire : Le partitionnement de la mémoire élargit la "largeur" de la hiérarchie de la mémoire en divisant, par duplication externe, un niveau de hiérarchie spécifique un présent au nombre de niveaux hiérarchiques. Cette résolution ne signifie pas seulement l'ajout pur et simple de niveaux supplémentaires de mémoires caches.

— Techniques axées sur les performances : Les résolutions de conception visant explicitement à optimiser les performances peuvent également contribuer à réduire l'énergie. Les routines qui tentent de diminuer la fréquence des ratés du cache par une conception appropriée du cache sont des exemples de telles solutions ; en fait, la réduction des ratés du cache diminue le coût pratique ordinaire de l'entrée des données dans la mémoire.

— Optimisation de la bande passante : La bande passante est décrite comme l'amplitude de l'interface processeur-mémoire (PMI), par opposition à la latence de la mémoire, qui est une force de la profondeur de la PMI. La bande passante de la mémoire devient de plus en plus importante en tant que métrique pour les systèmes modernes en raison du parallélisme accru au niveau des instructions généré par les processeurs super-scalaires ou VLIW, et en raison de la densité d'intégration qui permet des latences plus petites.

Banking a été reconnue comme l'une des meilleures techniques permettant de réduire la consommation d'énergie de la mémoire. Une nouvelle approche a été proposée pour améliorer l'efficacité de l'énergie de l'architecture de banc mémoire , en effectuant d'autres calculs, ce qui rend inutile la réactivation de la banc qui est en mode de fonctionnement à faible consommation d'énergie. Les modes de fonctionnement à faible consommation

d'énergie ont été utilisés pour améliorer la consommation d'énergie de l'architecture de mémoire en banc. Le fait de mettre les bancs de mémoire inactifs en mode de fonctionnement à faible consommation d'énergie permettrait de réduire la consommation d'énergie. Un nouveau système d'économie d'énergie est proposé sur le recalculé afin d'amplifier l'économie d'énergie, qui pourrait être réalisée en utilisant des systèmes de mémoire multibancs [117].

Les chercheurs ont discuté de l'aspect architecte dans les systèmes pour résoudre de nombreux problèmes sur les systèmes embarqués afin d'améliorer l'OMI et de réduire la consommation d'énergie, par : Direct-Mapped Cache Memories, Central Loop Buffer Architectures for Single Processor Organization, Multiple Loop Buffers Architectures with Shared Loop-Nest Organization, Distributed Loop Buffer Architectures with Incompatible Loop-Nest Organization and Instruction Fetch and Decode Improvements [11].

Les systèmes de mémoire à faible consommation d'énergie sont en cours de création et prennent en charge différents états de consommation des bancs de mémoire. Dans les états de faible consommation, la dissipation d'énergie est diminuée, mais le temps d'accès à la mémoire est amélioré. Il existe des techniques algorithmiques permettant de réduire l'énergie de la mémoire en réduisant le trafic de données et en concevant des programmes de gestion de l'énergie adaptés. On y parvient en examinant la structure d'un algorithme traité et le guide d'accès aux données pour concevoir des calendriers de gestion de l'énergie de la mémoire [197].

**Scratchpad** (SPM) est le plus en plus présentes dans les systèmes embarqués en raison de leur compétence plus incroyable en matière d'énergie et d'espace silicium que les caches ordinaires. Malgré cela, pour utiliser toutes ses ressources, des mécanismes d'allocation de mémoire efficaces doivent être produits.

Des études récentes sur la gestion de la mémoire ont considéré les mémoires de type scratchpad (SPM) gérées par logiciel comme un substitut aux hiérarchies basées sur les caches. Alors que les SPM sur puce ont des caractéristiques de performance/puissance similaires à celles des caches sur puce, leurs performances sont moins surprenantes et elles peuvent être plus efficaces sur le plan énergétique étant donné que définir les données sous le contrôle du programme [144]. En ce qui concerne les applications embarquées, les mémoires de type "scratchpad" (SPM) semblent être la meilleure solution de compromis

si l'on considère la consommation d'énergie, les performances et la surface de la puce. Le principal défi de la conception des SPM consiste à mettre parfaitement en correspondance les emplacements mémoire et les emplacements du scratchpad [9].

Utilisation de SPM pour remplacer le cache, le cache nécessite plus d'espace que les SPM (cet espace augmente la consommation d'énergie), car ces mémoires utilisent des tags pour savoir si les données sont sauvegardées dans le cache, elles doivent être mises en cache, ce qui nécessite plus de surface de puce pour stocker ces tags [16].

Les travaux qui utilisent les SPM pour l'optimisation de l'énergie dans les systèmes embarqués sont présentés suivant : [103, 106, 144]. Les auteurs de ces travaux ont travaillé sur les mémoires de type Scratch-Pad (SPM) plutôt que sur les caches. Ces articles proposent une perspective de gestion de la mémoire à l'exécution pour les SPM au niveau du système d'exploitation qui peut être fusionnée avec d'autres approches à la compilation. Ils ont également proposé de nouvelles heuristiques hybrides pour réduire la consommation d'énergie de la mémoire dans les systèmes embarqués, qui sont plus efficaces. Dans leur étude Kandemir *et al.* [111] ont présenté une stratégie d'optimisation de l'énergie de fuite basée sur un compilateur pour les mémoires de type scratch-pad (SPM) sur puce.

La combinaison de SPM dans l'architecture de la mémoire est une technique très courante dans la gestion de la mémoire pour réduire la consommation d'énergie [161].

Une utilisation adéquate de l'espace mémoire sur la puce est grandement nécessaire pour les applications des systèmes embarqués modernes basés sur des cœurs de processeurs. Il existe une technique pour utiliser efficacement la mémoire Scratch-Pad sur puce en partitionnant les variables scalaires et matricielles de l'application dans la DRAM hors puce et la SRAM sur puce, dans le but de minimiser le temps d'exécution total des applications embarquées [159].

Une méthode a été proposée pour améliorer l'efficacité de l'organisation des SPM dans la hiérarchie de la mémoire en suivant le comportement d'accès dynamique à la mémoire dû aux modèles de données d'entrée pour exécuter un accès irrégulier à la mémoire et/ou un flux de contrôle différent dans les applications [41]. L'optimisation basée sur la définition optimale de la hiérarchie de la mémoire, la fragmentation de la mémoire et l'allocation de données (variables locales et globales) est abordée dans la hiérarchie de la mémoire.

Comme décrit dans le travail de recherche de Angiolini *et al.* [9], ce travail montre un profilage d'application qui est réalisé par l'intention d'examiner les différentes sections de mémoire utilisées (pile, tas, code,etc). Ensuite, les sections de code et de données sont divisées en autant de SPMs que nécessaire (le nombre de SPMs dépend de sa taille qui est décidée a priori). Il existe de nombreux travaux sur ce sujet qui ont été soigneusement étudiés, par exemple : [67, 175, 216, 217].

Il existe des algorithmes conçus pour la gestion de diverses données statiques ou dynamiques pour SPM. Pour gérer efficacement le remplacement des données, les programmes sont divisés en zones selon les méthodes à Udayakumaran *et al.* [217], Ils ont utilisé l'algorithme, qui est conçu pour l'architecture pure SRAM-SPM.

Les différentes sections de la mémoire, telles que la SRAM en scratch-pad, la DRAM interne, la DRAM externe et la ROM, sont visibles directement par le logiciel, la gestion externe étant assurée par un mécanisme de cache matériel, gestion programmée par une mécanique de mise en cache matérielle. Pour saisir des données à l'aide de SPM, il existe des méthodes dynamiques ou statiques, parmi les méthodes statiques, nous disposons de ce travail d' Avissar *et al.* [14] qui a présenté une méthode de compilation permettant d'allouer les données du programme entre les différentes unités de mémoire des processeurs embarqués ne comportant pas de matériel de mise en cache, sans mise en cache, la tâche d'allocation des données aux différents bancs incombe au logiciel. Voir aussi d'autres méthodes statiques dans [16, 204, 220]. Pour les applications dynamiques, voir [169], ce travail propose une approche logicielle/matérielle embarquée pour la gestion du Scratchpad en temps réel. Pour plus d'informations dans ce contexte, voir : [41, 107, 221].

Aussi dans ce travail [190], il a y deux profileurs de mémoire pour le schéma de langage de programmation (KPROF+KBDB), ces deux outils sont appliqués sur le schéma de compilateur.

**Utilisation de la mémoire à changement de phase (PCM)** pour l'optimisation énergétique des systèmes embarqués. La PCM est considérée comme un choix sûr de DRAM (la PCM est utilisée pour remplacer la DRAM). Dans ce travail Shao *et al.* [193], ils ont examiné comment utiliser la PCM pour l'optimisation énergétique dans les systèmes embarqués. Ils ont présenté une architecture de système de mémoire hybride spécifique à une application dans laquelle le PCM est utilisé pour remplacer la DRAM

autant que possible. L'un des inconvénients de cet article est que l'optimisation de l'énergie avec le PCM dans les systèmes embarqués n'a pas été complètement discutée.

Il existe plusieurs techniques pour aborder l'optimisation matérielle. Par exemple dans Balasa *et al.* [15], ils ont à proposé une perspective non-scalaire pour le calcul de la taille de la mémoire pour les applications télécom et multimédia à dominante de données, où le stockage de grands signaux multidimensionnels entraîne un coût notable en termes de surface et de consommation d'énergie.

Les systèmes de mémoire à faible consommation d'énergie sont conçus pour maintenir les bancs de mémoire dans différents états de consommation. Dans les états de faible puissance, la consommation d'énergie est réduite, mais le temps d'accès à la mémoire est augmenté [197].

## 3.4 Gestion de mémoire orientée-logiciel (optimisation logicielle)

Dans cette section, nous présenterons les solutions logicielles disponibles qui aident les fabricants à construire une machine puissante fonctionnant avec des systèmes embarqués avec une faible consommation d'énergie, qui est considérée comme l'un des plus grands obstacles dans le monde de l'industrie. Habituellement, un système embarqué est constitué d'unités hétérogènes, parmi lesquelles la DRAM interne, la DRAM externe et la ROM telle que le scratchpad. Ces unités contrôlées directement par le logiciel, avec une gestion automatique des caches par le matériel, sont strictement déconseillées en raison de leur coût élevé et de l'utilisation excessive de la consommation énergétique ; cette manipulation des unités hétérogènes entraîne également de nombreuses difficultés et diminue les performances des systèmes en temps réel. Par conséquent, l'allocation des données à différentes unités de mémoire garantira la meilleure performance des puces, ce qui sera expliqué dans les sections ci-dessous.

**Schéma d'allocation optimale de la mémoire** (exemple scratch pad) [14] : Il existe d'autres méthodes pour augmenter les performances des systèmes embarqués, comme les algorithmes révolutionnaires multi-objectifs (MOEA), ils obtiennent une place de choix dans le monde de l'industrie en raison de leur brillante réputation d'optimiseurs de haut

niveau grâce à leur grand succès. Cependant, les MOEAs sont confrontés à de nombreux problèmes et bugs dans les applications implantées, car un grand nombre d'évaluations de performance est nécessaire (calcul objectif). En fait, il existe de nombreuses situations industrielles dans lesquelles les évaluations de la condition physique coûtent cher. Habituellement, les délais sont très courts ; ce type d'application doit également être créé selon une stratégie efficace à l'approche, qui doit être fournie. Ainsi, si l'on pense à un système embarqué complexe, il n'est pas évident de définir une architecture optimale. De plus, dans le temps central, en s'assurant qu'il sera prêt avant les délais. Cette phase du cycle de construction a reçu comme nom abrégé DSE (Design Space Exploration).

Comme nous l'avons mentionné plus haut, l'allocation optimale de la mémoire résout de nombreux problèmes concernant la réduction de la consommation d'énergie sur les systèmes embarqués. Cette méthode a été développée pour être utilisée dans les microcontrôleurs, et les processeurs DSP, comme le scratch-Pad SRAM, la logique derrière cette théorie est que le programmeur est libre de diviser les données entre les unités de mémoire existantes, cette méthode a prouvé son optimalité parce que l'allocation de mémoire a été profilée en temps réel et elle a détecté selon le profilage prédéfini parmi toutes les partitions existantes pour les données globales et de pile. En fait, cette méthode a permis aux programmeurs de distribuer les données entre les différentes unités, et elle évite strictement l'utilisation de la mise en cache matérielle qui n'est pas du tout recommandée dans les systèmes embarqués. De plus, nous ne devons pas ignorer la grande optimalité fournie par cette méthode, en raison de ces avantages, cette méthodologie est demandée pour les raisons suivantes :

— Cette méthode peut fonctionner avec seulement 20% des données de la SRAM.

— Il peut augmenter les performances du système de 56% en phase d'exécution de manière automatique, c'est-à-dire sans l'intervention d'un programmeur.

— En conséquence de l'utilisation de la stratégie de pile distribuée à la place de la pile unifiée, nos marques de branche montrent une réduction de 44,2% sur le temps réel.

— Selon l'utilisation du programme ou de l'application, le compilateur gère l'allocation de la mémoire de manière que le système utilise plus ou moins de mémoire en fonction du nombre d'accès aux données, c'est-à-dire qu'elle change relativement et arbitrairement avec l'accès aux données.

Et c'est l'un des défis les plus importants pour l'EDA (Electronic Design Automation). Plus tard dans cette étude **Évaluation des performances de l'Efficient Multi-Objectif** [12], nous approfondirons cette technologie très demandée. De nombreuses autres technologies ont fait un grand pas pour résoudre ce problème commun.

Aujourd'hui, le processus de conception des systèmes informatiques ou de l'architecture des processeurs a changé ; il ne s'agit plus seulement de rapidité, mais aussi d'un grand défi pour améliorer l'optimalité des systèmes. Les systèmes embarqués sont l'un des plus concernés, notamment par la consommation d'énergie. Les MOEA sont utilisés pour configurer tous les paramètres. En d'autres termes, la méthode trace tous les paramètres pour trouver les configurations quasi-optimales en peu de temps.

Une fois les paramètres fondés, le système redéfinit le modèle approximatif. De plus, ce processus passe par deux phases, d'abord il simule chaque configuration et ces résultats seront utilisés pour entraîner les systèmes flous à devenir fiables. Dans un deuxième temps, le système flou sera reconfiguré et affiné. Dans cette étape, le processeur entier sera conçu avec moins de problèmes et de questions.

Plus tard dans ce travail Ozturk *et al.* [157], **Utilisation de la compression des données pour augmenter la mémoire**, nous aurons une meilleure idée de la façon dont la diminution et le compactage des données peuvent donner un coup de pouce à notre système embarqué sur la réduction de la puissance de calcul, par le support automatisé du compilateur peut aider à décider de l'ensemble des éléments de données à compresser et décompresser en temps réel en fonction des besoins du système, pour garantir ses performances.

L'occupation de la mémoire est souvent le problème le plus critique de l'évolution des systèmes embarqués, tant au niveau de la conception que de l'optimisation. En particulier, l'augmentation du nombre de codes de micro-programmes et de la quantité de données traitées sont les facteurs les plus significatifs de cette problématique des systèmes multiprocesseurs.

Le multi-accès aux données sur un système multiprocesseur sur une puce (MPSoc) est une raison fondamentale de la consommation d'énergie. Le rapport entre les deux points.

Par exemple, l'exécution d'une boucle while oblige le système à accéder en permanence à ce qui se produit une forte consommation d'énergie. Cette étude vise à découvrir

comment nous pouvons automatiser le compilateur afin qu'il compresse/décompresse les données en fonction des besoins des processus pendant l'exécution de la boucle.

Dans un premier temps, l'étude passe par un système monoprocesseur. Ensuite, l'expérimentation sera étendue sur un système multi-processus. La proposition initiale était un algorithme comportant un schéma statique et un schéma dynamique. Sur la partie dynamique, l'exécution du programme est basée sur le regroupement, qui peut être modifié automatiquement pendant l'exécution selon le type de données et le modèle d'accès comme mentionné en haut.

Cette expérience montre que la méthode utilisée a réussi à réduire l'occupation de la mémoire à 47,9% et dans le meilleur des cas à 48,3%. De même, cette méthode a permis de fixer la moyenne de la sauvegarde dans MPSoc à 12,7%.

Son travail a permis de réduire l'occupation maximale et moyenne de la mémoire en insérant une fonction de compression et de décompression dans l'application pendant l'exécution du programme. Ce travail a ouvert la porte à de futures recherches sur la compression et la décompression des données pendant le traitement des données.

Pour plus de détails, voir la compression de données guidée par le compilateur pour réduire la consommation de mémoire des applications embarquées [156]. Comme mentionné dans cette étude, la complexité des codes est l'un des facteurs les plus courants de l'utilisation élevée de l'énergie, donc le chercheur a creusé plus profondément en présentant et en évaluant une approche pilotée par un compilateur pour simuler l'occupation de la mémoire. L'objet de cette étude est le même que l'expérience précédente : la création d'un compilateur qui décide lui-même comment définir les données en fonction des besoins du programme pendant l'exécution du processus et le résultat de ce travail a confirmé l'efficacité de la répartition des données en fonction des besoins du programme pour réduire le coût du calcul de l'énergie sur les systèmes embarqués.

Nous allons approfondir **la synthèse de mémoire basée sur la disposition pour les systèmes embarqués sur puce [22]**, qui avait offert de nouvelles possibilités de réduire l'énergie des systèmes en utilisant les modèles d'accès à la mémoire pour mappage l'utilisation de l'énergie en fonction des adresses les plus fréquemment consultées. Cette nouvelle méthode pourrait être une meilleure occasion de réduire la consommation d'énergie des systèmes embarqués. La logique derrière cette méthode est assez simple,

mais elle est efficace. L'avantage de cette méthode est que le programme peut profiler l'accès aux systèmes dès la phase de conception. La première étape consiste à mapper la plupart des adresses accédées sur la puce SRAM, ce qui permet de garantir l'efficacité énergétique et les performances. Dans ce travail, le chercheur a proposé un algorithme pour automatiser le partitionnement de la SRAM en différentes bancs. Cet algorithme est une solution optimale aux contraintes sur le nombre d'une banc de mémoire. Grâce à cette méthode, le chercheur a pu diminuer l'utilisation de l'énergie jusqu'à 34%.

En raison de l'importance du comportement des données et de son rapport avec la consommation d'énergie, nous allons expliquer ici l'étude **Organisation de la mémoire des données et accès aléatoire aux données** [145, 161], les meilleures architectures disponibles qui permettent de réduire la consommation d'énergie. Sur ces références, les chercheurs ont essayé de comprendre l'architecture de la conception et son association avec les données de l'utilisateur pourrait résoudre de nombreux problèmes sur les systèmes embarqués en créant un compilateur qui travaillera sur un mappage efficace des données pour offrir un environnement intelligent afin qu'ils adaptent l'organisation de la mémoire pendant la phase de mise en œuvre du système. Cette étude a permis de créer un flux qui organise et optimise l'accès aux données et l'allocation de mémoire, l'affectation et l'accès aux données. Cette optimisation se fait à l'aide d'un script qui inclut des techniques qui travaillent et détectent automatiquement la meilleure méthode pour organiser la mémoire et l'accès aux données afin d'atteindre un haut niveau de performance et une réduction maximale de la consommation de données.

Dans ce qui suit, nous verrons comment **fonctionne la compression de données assistée par le matériel pour la minimisation de l'énergie dans les systèmes avec processeurs embarqués**[21].

L'une des autres méthodes testées sur les systèmes embarqués est l'assistance matérielle à la compression/décompression des données comme solution à la consommation d'énergie.

Les chercheurs ont essayé de trouver une nouvelle architecture efficace pour la compression et la décompression des données à la demande, en recherchant le cache.

Cette étude a comparé le profil-driven et différentiel ; cette étude a comparé leur performance, et le trafic de mémoire et la consommation d'énergie dans le chemin de la

mémoire cache puisque la compression de l'information a une large utilisation dans les nouvelles technologies pour traiter la bande passante du bus.

Différents scénarios ont été proposés et testés pour évaluer les performances des systèmes après application de la compression et de la décompression. Fondamentalement, cette étape permet au chercheur de surveiller les comportements des conceptions au niveau matériel ; une fois que les données ont été stockées dans les mémoires et le cache, la compression/décompression vient résoudre le problème de la performance, car le matériel doit traiter les données à ce moment-là, ce qui fait de la vitesse de compression le facteur le plus important et le plus efficace de ce processus. Les solutions assistées par le matériel sont l'option la plus utilisée dans ce contexte.

En outre, le chercheur n'ignore pas la taille et la complexité du matériel informatique puisque le Scos est utilisé dans le processus de mise en œuvre. La raison de l'utilisation de la compression de mémoire assistée par le matériel informatique est qu'il a été prouvé que cette approche peut être utilisée lorsque l'objectif est la minimisation de l'énergie.

Cette étude a donc proposé la compression de données assistée par matériel comme solution performante pour réduire la consommation d'énergie sur les systèmes embarqués. En préparant un nouveau modèle architectural pour la compression des données, cet architecte a atteint l'objectif de réduire la consommation d'énergie de 4,2% à 35,2%, selon le schéma de compression utilisé.

Dans ce travail, **organisation de la mémoire d'instructions (IMO)** [11], présente une combinaison des procédures à faible consommation d'énergie qui sont utilisées dans l'organisation de la mémoire d'instructions (IMO). L'IMO étant considérée comme l'une des principales sources de consommation d'énergie dans les systèmes embarqués. Ce travail décrit également l'un des principaux types d'améliorations liées à l'aspect logiciel : Le profilage pour l'optimisation du code, les transformations du code source, le mappage.

## 3.5 Gestion de mémoire orientée-matériel/logiciel

De nombreuses études ont traité du problème de la liaison des données. À savoir, l'allocation d'une structure de données d'une application particulière à une structure de mémoire

unique. Dans ce travail Panda *et al.* [160], ils ont présenté une étude des approches actuelles et émergentes d'optimisation des données et de la mémoire pour les systèmes embarqués. Ils ont d'abord présenté les optimisations de la mémoire indépendantes de la plate-forme, qui s'exécutent au niveau de la source et garantissent généralement une amélioration des performances, de la puissance et des coûts, sans tenir compte de l'architecture cible de l'implémentation.

Le travail présenté dans cet article Artes *et al.* [11], donne une émulsion sur les méthodes à faible énergie qui sont utilisées dans les organisations de mémoire d'instructions, en définissant leurs avantages comparatifs, leurs inconvénients et leurs compromis.

Il a été prouvé que la prépondérance du coût en surface et en énergie n'est pas liée au chemin de données ou aux contrôleurs dans les systèmes qui concernent des flux multidimensionnels de signaux, comme les images ou les séquences vidéo, mais à la communication globale et à l'interaction avec la mémoire. En fait, 50% à 80% de la consommation d'énergie des circuits spécifiques aux applications (ASIC) pour le traitement du signal en temps réel est liée au trafic mémoire causé par les transferts entre l'ASIC et les mémoires hors puce. Cela suggère qu'avec une conception précise, la déduction du coût énergétique lié à la mémoire peut largement dépasser la réduction due à la mise à l'échelle de la tension et aux autres transfigurations permettant d'économiser de l'énergie [195].

De nombreuses études se sont intéressées au problème du partitionnement de la mémoire pour une énergie faible. Ces études considèrent le nombre et les capacités des bancs de mémoire et le nombre d'accès aux variables. D'autres travaux considèrent les contraintes précédemment citées et utilisent une mémoire externe.

**Problème du partitionnement de la mémoire pour une faible consommation d'énergie** :

Il existe quelques routines pour résoudre le problème de la division de la mémoire pour les faibles énergies.

Dans ces deux études [22] et [20], Ils ont proposé un algorithme pour le partitionnement automatique des SRAMs sur puce en plusieurs bancs. L'algorithme calcule une solution optimale au problème sous des hypothèses pragmatiques sur les métriques de coût énergétique et des contraintes sur le nombre de bancs de mémoire.

En Avissar *et al.* [14], les piles sont partitionnées en plusieurs bancs de mémoire et/ou hiérarchie de mémoire afin de diminuer le coût d'accès (le coût d'accès est défini par le temps nécessaire pour accéder à une donnée dans la mémoire et la consommation d'énergie associée) des variables locales et globales.

Une nouvelle méthode d'allocation et d'affectation de mémoire utilisant le partitionnement de la mémoire pour personnaliser l'architecture de la mémoire. Cependant, l'algorithme de partitionnement pour résoudre les conflits dans le graphe de conflit n'est pas décrit. Cet article identifie l'espace d'exploration approprié se situant entre deux minima, les nombres chromatiques acquis par l'algorithme de coloration et le nombre de nœuds. Le nombre de modules de mémoire à allouer ainsi que la taille et le type de chaque module de mémoire sont définis dans l'espace d'exploration [113].

Un algorithme de partitionnement à coupe minimale est utilisé pour l'allocation et l'affectation de la mémoire. Pour utiliser cet algorithme, le graphe de conflit est nécessaire et le concepteur doit définir un nombre de séparations (c'est-à-dire le nombre de bancs). En outre, l'algorithme min-cut a recours à l'obtention de coupes minimales dans le graphe de conflit, ne résolvant que les conflits minimaux. Le graphe de conflit est modifié de manière à maximiser les coupures. La maximisation de la coupure entraîne la résolution du nombre maximal de conflits dans le graphe de conflit[112].

Une technique de partitionnement automatique de la mémoire pour l'optimisation du débit et de la puissance. L'optimisation du débit vise à maximiser les accès synchrones à la mémoire par le partitionnement ; par ailleurs, l'optimisation de la puissance vise à fermer les bancs de mémoire inutilisés pour économiser la consommation d'énergie dynamique. Cette technique utilise un algorithme de branchement et de délimitation pour examiner la meilleure combinaison de séparations [48].

Ils gardent à l'esprit et prennent en compte les capacités des bancs de mémoire, leurs tailles et le nombre d'accès aux variables pour discuter du problème de la réduction du nombre de bancs de mémoire simultanément actifs afin que les autres bancs de mémoire qui sont inefficaces puissent être placés dans des modes de faible puissance pour réduire la consommation d'énergie. L'architecture considérée comporte plusieurs bancs de mémoire et de nombreux modes de fonctionnement à faible consommation pour chacun de ces bancs. Ce problème est modélisé comme un problème de partitionnement de graphe à plusieurs voies et des heuristiques bien connues sont utilisées pour le résoudre [196].

**Utilisation de la mémoire externe :**

L'objectif principal d'un dispositif de stockage est de mémoriser quelques mots de données de n bits pour une courte ou longue durée. La littérature sur le stockage fait état de nombreux types possibles de RAM à utiliser comme mémoires principales et la recherche sur la technologie RAM est toujours très active. Les SRAM sont éliminées pour les grandes tailles nécessaires aux mémoires prévalentes en raison de leur densité nettement inférieure, de plus d'un ordre de grandeur, à celle des (S)DRAM. De nos jours, les rendements multimédias riches en fonctionnalités nécessitent le résultat d'un système embarqué avec un système sur puce (SoC) composé pour répondre aux attentes du marché en matière de hautes performances à faible coût et de consommation d'énergie réduite [145].

Certains travaux utilisant la mémoire externe, tels que ceux apparaissant dans [118], ont abordé le problème de l'exploration multi-niveau multi-objectif de l'architecture mémoire à travers une fusion de l'algorithme génétique multi-objectif (exploration de l'architecture mémoire) et d'un algorithme heuristique efficace de placement des données. L'exploration de l'architecture mémoire est effectuée au niveau extérieur en prenant des modules mémoire directement dans une bibliothèque mémoire ASIC.

## 3.6 Conclusion

Dans les systèmes embarqués, la gestion de la mémoire a un impact essentiel et direct sur les métriques des coûts globales (surface, performance, énergie, etc.). En d'autres termes, l'optimisation de cette allocation conduit à des gains substantiels en termes de temps d'exécution et de consommation énergétique.

Dans le chapitre suivant, nous examinerons un problème d'allocation de mémoire dans les systèmes embarqués, nous verrons comment les chercheurs ont pu fournir des modèles en utilisant les mathématiques et comment ils ont proposé des algorithmes pour réduire la consommation d'énergie, ce qui est l'un des défis les plus importants de la création d'un système embarqué.

# Chapitre 4

# Allocation de mémoire dans un système embarqué

## 4.1 Introduction

La mémoire (type, taille, taille de l'application et performances qu'elle contient) a un impact direct et essentiel sur la consommation d'énergie et les performances. Pour réduire la consommation d'énergie, il faut déterminer l'allocation optimale de la mémoire.

La mémoire est bottleneck des performances des architectures modernes[68, 190]. La gestion de la mémoire est donc une partie essentielle d'un système embarqué actuel. Le problème L'allocation de mémoire est et devient une partie nécessaire des systèmes embarqués d'aujourd'hui. Il s'agit d'un problème académique dans la science embarquée en entraînant une certaine complexité. La résolution du problème d'allocation de mémoire minimise le coût de la mémoire. Cette solution est un art de traiter efficacement la mémoire des systèmes embarqués et une bonne gestion. L'objectif le plus important est de décrire une hiérarchie optimale de la mémoire, la segmentation de la mémoire et l'allocation des données (variables locales et globales).

Un problème d'allocation de mémoire peut être modélisé comme une version du bin packing (Le bin packin est un problème algorithmique, a été récemment introduit par Chung et al [45]), où les objets peuvent être divisés, mais où chaque bin peut contenir au maximum deux (parties d'objets).

Mais il n'est pas facile d'estimer l'utilisation de la mémoire des programmes implémentés dans les langages fonctionnels, en raison à la fois de la traduction complexe de certaines constructions de haut niveau, et de l'utilisation de gestionnaires de mémoire automatiques. Pour aider à comprendre le comportement d'allocation de mémoire des programmes Schéma, il y a deux outils complémentaires. Le premier rend compte de la fréquence d'allocation, de la congestion du tas et de la récupération de la mémoire. Le second traque les fuites de mémoire, ces outils ont été mis en place appliquée au schéma de compilateur [68].

Nous étudions le problème de l'allocation des structures de données en mémoire pour réduire la consommation d'énergie. Ce dernier est mobilisé comme un problème d'optimisation combinatoire. En microélectronique, ce problème complexe est peu abordé, car les concepteurs ne maîtrisent pas les outils mathématiques de l'optimisation discrète. L'approche retenue, dans la mesure du possible, est un algorithme exact qui permet

d'obtenir une solution optimale. Une méthode alternative serait de proposer une bonne méta-heuristique.

La résolution consiste à utiliser au mieux la petite mémoire partagée, car l'espace mémoire peut être divisé en bancs de mémoire, et chacun d'entre eux peut être contrôlé indépendamment. Il est également possible d'utiliser une mémoire principale externe, à partir de laquelle les données nécessaires sont transférées vers les bancs de mémoire selon les besoins. Le processeur accède aux données des bancs de mémoire et de la mémoire principale. Selon la structure proposée par [200] (voir le figure 4.1 [201]), les données d'application peuvent être stockées dans des bancs de mémoire directement et sans la présence de la mémoire principale. Aussi les données peuvent se trouver à l'intérieur de la mémoire principale où les données pertinentes sont transférées vers les bancs de mémoire, si elles sont essentielles et le processeur y accède en une seule fois avec certaines restrictions. Il existe une possibilité de créer un conflit fermé ou ouvert. Cela se traduit donc par un coût supplémentaire en cas de conflit, lorsque le processeur accède à la mémoire principale si elle existe et lorsqu'il accède également aux structures de données dans les bancs de mémoire.

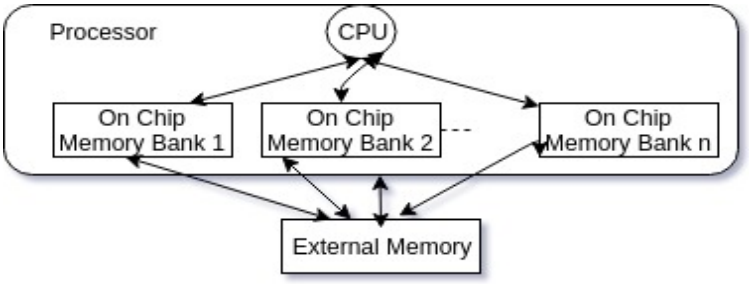


FIGURE 4.1 – Architecture générique utilisée

Dans cette section, nous décrivons certains problèmes d'allocation de mémoire dans les systèmes embarqués. Plusieurs modèles pour ce problème et des propositions d'optimisation issues de la recherche opérationnelle pour l'aborder. Dans ces travaux [200, 201] ils abordent différentes versions du problème d'allocation de mémoire. Ces recherches ont d'abord introduit l'architecture générique (voir le figure 4.1). L'architecture cible utilisée (c'est-à-dire le processeur et les bancs de mémoire, la mémoire externe). Le flux estimé (extraction des paramètres) sera proposé et essentiellement la détermination du graphe de conflit en mémoire, qui permet de mesurer le coût de chaque accès mémoire.

Les versions statiques de l'allocation de mémoire avec un degré de complexité différent introduites par Soto *et al.* [200]. Ces versions traitent quatre problèmes d'allocation de mémoire :

## 4.2 Allocation de mémoire sans contrainte

Ce problème est lié à la première version des techniques d'optimisation matérielle abordées dans ce chapitre (3) (voir la sous-section 3.3). Il s'agit de trouver le nombre minimum de bancs de mémoire dont une application particulière a besoin. En d'autres termes, elle ne cherche pas à allouer de la mémoire pour les structures de données.

La consommation d'énergie diminue lorsque la banc de mémoire est petite. Les concepteurs de circuits doivent donc constamment trouver un équilibre entre le coût de la structure de la mémoire et la consommation d'énergie. Au sens général de la taille et du nombre de bancs de mémoire, et de la consommation d'énergie. Par conséquent, les concepteurs de circuits veulent déterminer un compromis entre le coût de l'architecture et la consommation d'énergie. La charge du matériel double avec l'augmentation du nombre de bancs à cause de l'accroissement des sources de communication nécessaires à la transmission des informations et par la duplication de la logique d'adressage et de contrôle. Obtenir une partition optimale avec des limitations strictes sur le nombre maximum de bancs de mémoire est parce qu'il y a une augmentation des coûts dans l'augmentation de puissance, le cycle de temps et l'espace qui restreint le partitionnement arbitraire. La consommation d'énergie n'est pas coûteuse quand la taille et le nombre de bancs de mémoire sont limités, mais quand le nombre de bancs de mémoire augmente, la consommation augmente le coût, car le recyclage et la logique sont fréquemment recyclés et quand les ressources sont utilisées pour transférer les données[22].

### 4.2.1 Description du problème

Les structures de données ne peuvent pas être divisées et étendues sur plusieurs bancs de mémoire. De plus, la structure de données n'est pas en conflit avec une autre structure de données ; autrement dit l'application pourrait avoir des structures de données isolées. Par ailleurs, si ses structures de données sont allouées à la même banc de mémoire, un

conflit ouvert lui est dit. Sinon, il est dit conflit fermé en considérant que le processeur peut accéder simultanément à tous ses bancs de mémoire.

Le problème d'allocation de mémoire sans contrainte est compris comme suit : pour une application assignée, il faut examiner le nombre minimum de bancs de mémoire pour lesquels tous les conflits non automatiques sont fermés. En détail, un auto-conflit est toujours ouvert, et il n'est alors pas possible de trouver une résolution des conflits disponibles à l'extérieur.

### 4.2.2 Formulation mathématique

Une formulation PLNE pour le problème d'allocation de mémoire sans contrainte : dans cette formulation PLNE, le nombre de structures de données est utilisé comme une borne supérieure du nombre de bancs de mémoire.

Le nombre de structures de données est indiqué par $n$. Le nombre de conflits est indiqué par $o$. Le conflit $k$ est modélisé comme la paire $(k_1, k_2)$, où $k_1$ et $k_2$ sont deux structures de données conflictuelles.

Les variables ont été représentées sous la forme d'une matrice binaire $X$. Il s'agit des variables de décision liée au problème : affectation des structures de données aux bancs de mémoire :

$$x_{ij} = \begin{cases} 1 & \text{si la structure de données } i \text{ est allouée à} \\ & \text{la banc de mémoire } j \quad \forall\, i, j \in \{1, \cdots n\} \\ 0 & \text{autrement} \end{cases}$$

Le vecteur des variables réelles non négatives $Z$ représente également la banc de mémoire qui est effectivement utilisée :

$$z_j = \begin{cases} 1 & \text{si au moins une structure de données est affectée} \\ & \text{à la banc de mémoire} \quad \forall\, i, j \in \{1, \cdots n\} \\ 0 & \text{autrement} \end{cases}$$

Enfin, le programme linéaire en nombres entiers (PLNE) est :

$$\text{Minimize} \quad \sum_{j=1}^{n} z_j \tag{4.1}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \forall\ i \in \{1, ..., n\}, \tag{4.2}$$

$$x_{k_1,j} + x_{k_2,j} \leq 1, \forall k_1 \neq k_2, \forall j \in \{1, ..., n\}, \forall\ k \in \{1, ..., o\}, \tag{4.3}$$

$$x_{i,j} \leq z_j, \forall i, j \in \{1, ..., n\}, \tag{4.4}$$

$$x_{i,j} \in \{0, 1\}, \forall (i, j) \in \{1, ..., n\}^2, \tag{4.5}$$

$$z_j \geq 0, \forall j \in \{1, ..., n\}. \tag{4.6}$$

La fonction objectif du problème minimise le nombre de bancs de mémoire utilisés pour stocker les structures de données de l'application (voir 4.1).

Le problème de l'allocation de mémoire sans contrainte est égal à la détermination du nombre chromatique d'un graphique de conflit Soto *et al.* [198].

### 4.2.3 Méthodes de résolution

Cette version du problème d'allocation de mémoire peut être considérée comme le problème de coloration de graphes à k-poids [34]. La formulation PLNE est résolue avec Xpress-MP. Puis, il y a deux méta-heuristiques proposées (Evo-Allocation basée sur un algorithme évolutionnaire hybride et Tabu-Allocation basée sur la méthode de recherche tabou). Ces deux méta-heuristiques sont inspirées des algorithmes du problème de coloration des sommets. Ensuite, une comparaison entre les résultats expérimentaux des méta-heuristiques et la formulation exacte résolue (testée sur des instances plus ou moins grandes [170]). Les solutions trouvées sont de très bonne qualité, mais la solution optimale est inconnue. Les résultats suggèrent que les méthodes de coloration de graphes peuvent être étendues avec succès à des problèmes d'allocation de mémoire plus complexes dans les systèmes embarqués [200].

## 4.3 Allocation de mémoire avec contraintes sur le nombre de bancs de mémoire

Ce problème, est la deuxième version liée aux problèmes de liaison de données introduits dans le chapiter 3 (voir sous section 3.5).

L'obtention d'une allocation de mémoire optimale pour les structures de données avec une contrainte sur le nombre maximum de bancs de mémoire est très importante. Pour des raisons de coût et de conception, le nombre de bancs de mémoire disponibles est limité. Il est choisi au préalable par le concepteur. En outre, lorsque le nombre de bancs augmente, les ressources de communication nécessaires pour transférer les informations et la logique de contrôle augmentent en même temps [22].

### 4.3.1 Description du problème

Le problème a pour but d'allouer les structures de données d'une application donnée, à une architecture mémoire donnée. La spécificité de l'architecture mémoire est que le nombre de bancs de mémoire est fixe.

Le processeur entre simultanément dans tous les bancs de mémoire, bien que chaque conflit ait un coût (en ms). Ce coût de conflit est comparable au nombre de fois qu'un conflit d'application se produit. Les situations de conflit, ouvert et fermé, sont définies comme suit : D'abord, un conflit fermé s'ils ont deux structures de données qui sont allouées à deux bancs de mémoire différents et le conflit n'engendre aucun coût. Le conflit ouvert s'ils ont deux structures de données qui sont allouées à dans la même banc et le conflit génère un coût $d_k$.

Dans certains états, il n'est pas simple de mesurer le coût des conflits. Dans cet état, les outils de profilage de code peuvent être utilisés pour l'estimation des coûts des conflits sur une base statistique [124].

Pour ce problème de deuxième version, une vérification de l'affectation d'un certain nombre de bancs de mémoire à des structures de données, de sorte que le coût total des conflits résultant des conflits ouverts soient minimisé.

### 4.3.2 Formulation mathématique

Pour résoudre ce problème, Soto *et al.* [200], ils ont proposé une formulation PLNE. Le nombre de structures de données est indiqué par $n$, le nombre de conflits indiqué par $o$, et un conflit $k$ est modélisé comme la paire $(k_1, k_2)$, où $k_1$ et $k_2$ sont deux structures de données.

Ce problème tient compte d'un nombre fixe de bancs de mémoire désigné par $m$. Les coûts de conflit associés aux conflits, qui sont désignés par $d_k$ pour tous $k \in \{1, \cdots o\}$.

Il existe deux vecteurs de variables de décision, le premier, c'est la matrice binaire $X$, où :

$$x_{ij} = \begin{cases} 1 & \text{si la structure de données } i \text{ est attribuée à la banc} \\ & \text{de mémoire } j \text{ pour tous } i \in \{1, \cdots n\} \text{ et pour tous } j \in \{1, \cdots m\} \\ 0 & \text{autrement} \end{cases}$$

La deuxième, est un vecteur des variables réelles non négatives $Y$, où :

$$y_k = \begin{cases} 1 & \text{si le conflit } k \text{ est ouvert pour tous } k \in \{1, \cdots o\}. \\ 0 & \text{autrement} \end{cases}$$

Enfin, le programme linéaire en nombre entier (PLNE) pour cette version du problème d'allocation de mémoire est :

$$\text{Minimize} \quad \sum_{k=1}^{o} y_k d_k \quad , \tag{4.7}$$

$$\sum_{j=1}^{m} x_{i,j} \quad = \quad 1, \forall\ i \in \{1, ..., n\}, \tag{4.8}$$

$$x_{k_1,j} + x_{k_2,j} \quad \leq \quad 1 + y_k, \forall j \in \{1, ..., m\}, \forall\ k \in \{1, ..., o\}, \tag{4.9}$$

$$x_{i,j} \quad \in \quad \{0, 1\}, \forall (i, j) \in \{1, ..., n\} * \{1, ..., m\}, \tag{4.10}$$

$$y_k \quad \geq \quad 0, \forall k \in \{1, ..., o\}. \tag{4.11}$$

La fonction de coût (4.7) est égal à la somme totale des coûts des conflits ouverts. La contrainte (4.8) chaque structure de données est assignée à une seule banc de mémoire. La contrainte (4.9), définit la variable $y_k$ à sa valeur appropriée.
$y_k$ est égal à 1 si le conflit $k$ implique une structure de données en conflit avec lui-même ($k_1 = k_2$). La contrainte (4.10), en force intégrabilité contraint on $x_{i,j}$. Enfin, La contrainte (4.11), définit $y_j$ comme variable non négative pour tout $j$ [200].

### 4.3.3 Méthodes de résolution

Le problème d'allocation de mémoire avec contrainte sur le nombre de bancs de mémoire est également équivalent au problème de coloration de graphe à k-poids [34]. La formulation PLNE résolue avec Xpress-MP. Deux méta-heuristiques proposées (Evo-Allocation basée sur un algorithme évolutif hybride et Tabu-Allocation basée sur la méthode de recherche tabu). Ces deux méta-heuristiques sont inspirées des algorithmes du problème de coloration des sommets. Puis une comparaison entre les résultats expérimentaux des méta-heuristiques et la formulation exacte résolue (testé sur des instances plus ou moins importantes [170]). Les solutions trouvées sont de très bonne qualité, mais la solution optimale est inconnue. Les résultats suggèrent que les méthodes de coloration de graphes peuvent être étendues avec succès à des problèmes d'allocation de mémoire plus complexes dans les systèmes embarqués [200].

## 4.4 Allocation de mémoire générale

La troisième version du problème d'allocation de mémoire (MemExplorer), est lié aux problèmes de liaison de données qui est présenté dans le chapitre 3 (sous la section 3.5).

Par rapport à la deuxième version 4.3, il existe désormais une mémoire externe dans l'architecture cible, et également des contraintes supplémentaires sur les bancs de mémoire et les structures de données.

Les mémoires externes stockent les données à long terme et améliorent la productivité d'un système intégré. L'objectif général est d'allouer les structures de données d'une application particulière à une structure de mémoire particulière [145].

### 4.4.1 Description du problème

Pour ce problème, les capacités des bancs de mémoire sont limitées et le nombre de bancs de mémoire est fixe. La taille des structures de données et le nombre d'accès à celles-ci sont tous deux pris en compte. En outre, le temps d'accès à la mémoire externe est également pris en compte. Pour les conflits, le coût est créé si le conflit est ouvert. Aucun coût n'est créé si le conflit est fermé. Tous les bancs de mémoire sont accessibles simultanément.

Le MemExplorer se présente comme suit : pour un certain nombre de bancs de mémoire et de mémoire externe, l'allocation de mémoire sera recherchée pour les structures de données de sorte que le temps passé à accéder à ces données soit minimisé [200].

### 4.4.2 Formulation Mathématique

Le nombre de bancs de mémoire est indiqué par $m$, et le banc de mémoire $m+1$ fait référence à la mémoire externe. La capacité de la banc de mémoire $j$ est $c_j$ pour tous $j \in \{1, \cdots m\}$ (il est rappelé que la mémoire externe pas liée à une contrainte de capacité). Le nombre des structures de données est indiqué par $n$. La taille d'une structure de données $i$ est indiqué par $s_i$ pour tous $i \in \{1, \cdots n\}$. Outre sa taille, chaque structure de données $i$ est également caractérisée par le nombre de fois que le processeur y accède, elle est dénotée par $t_i$ pour tous $i \in \{1, \cdots n\}$. $t_i$ représente le temps nécessaire pour accéder à la structure de données $i$ si elle est allouée à une banc de mémoire. Si une structure de données $i$ est mise en correspondance avec la mémoire externe, son temps d'accès est égal à $p * t_i$ . Le conflit $k$ est associé à son coût $d_k$, pour tous les $k \in \{1, \cdots o\}$, où $o$ est le nombre de conflits [200].

Comme nous l'avons vu dans les versions précédentes, il existe deux variables de décision, la première représente l'affectation des structures de données aux bancs de mémoire. Ces variables sont modélisées sous la forme d'une matrice binaire $X$ :

$$x_{ij} = \begin{cases} 1 & \text{si la structure de données } i \text{ est allouée à la banc de mémoire } j \\ 0 & \text{autrement} \end{cases}$$

La deuxième, est un vecteur de variables réelles non négatives $Y$ , qui modélise les statuts de conflit ; ainsi la variable $y_k$ associée au conflit $k$ a deux valeurs possibles :

$$y_k = \begin{cases} 1 & \text{si le conflit } k \text{ est fermé} \\ 0 & \text{autrement} \end{cases}$$

Le modèle mathématique pour MemExplorer est représenté comme suit [200] :

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{m} t_i x_{i,j} \quad + \quad p\sum_{i=1}^{n} t_i x_{i,m+1} - \sum_{k=1}^{o} y_k d_k \tag{4.12}$$

$$\sum_{j=1}^{m+1} x_{i,j} \quad = \quad 1, \forall\ i \in \{1, ..., n\} \tag{4.13}$$

$$\sum_{i=1}^{n} x_{i,j} s_i \quad \leq \quad c_j, \forall\ j \in \{1, ..., m\} \tag{4.14}$$

$$x_{k_1,j} + x_{k_2,j} \quad \leq \quad 2 - y_k, \forall j \in \{1, ..., m+1\}, \forall\ k \in \{1, ..., o\} \tag{4.15}$$

$$x_{i,j} \quad \in \quad \{0, 1\},\ \forall (i, j) \in \{1, ..., n\} \times \{1, ..., m\} \tag{4.16}$$

$$y_k \quad \geq \quad 0,\ \forall k \in \{1, ..., o\}. \tag{4.17}$$

La fonction de coût (4.12) est le temps total (exprimé en ms) nécessaire pour accéder à toutes les structures de données. La contrainte (4.13) exprime que chaque structure de données est affectée à une seule banc de mémoire (qui peut être la mémoire externe). La contrainte (4.14) spécifie que la taille totale des structures de données affectées à la banc mémoire $i$ ne doit pas dépasser sa capacité. La contrainte de type (4.15) traite les conflits entre structures de données, si $y_k = 0$ alors la contrainte est redondante, car le conflit est ouvert tandis que $y_k = 1$ interdit d'allouer les deux structures de données à la même banc de mémoire. Enfin, toutes les variables sont binaires.

Les structures de données isolées et l'état dans lequel une structure de données est en conflit avec elle-même sont tous deux considérés en considération dans un de la formulation PLNE et de la méta-heuristique proposée Soto *et al.* [200].

### 4.4.3 Méthodes de résolution

Une approche exacte et une méta-heuristique basée sur VNS sont proposées pour résoudre le problème de MemExplorer. La méta-heuristique Vns-Ts-MemExplorer obtient des résultats encourageants pour résoudre le problème MemExplorer grâce à sa recherche bien équilibrée (intensification/diversification) [199, 200]. Ces méthodes des résolution testé sur une ensemble des instances [170].

## 4.5 Allocation dynamique de mémoire

La quatrième version du problème d'allocation de mémoire (MemExplorer-Dynamic), est liée aux problèmes de liaison de données qui ont été présentés dans le chapitre 3 (sous section 3.5).

L'objectif est d'allouer les structures de données d'une application donnée à un ensemble donné de bancs de mémoire. Dans cette variante, le temps d'exécution est divisé en périodes de temps. L'allocation de la mémoire prend en compte le besoin et les contraintes de chaque période. Ainsi, l'allocation de mémoire n'est pas statique, elle peut être améliorée puisque les demandes de structures de données des applications peuvent changer à chaque période [200].

### 4.5.1 Description du problème

La principale différence entre MemExplorer et cette version dynamique du problème d'allocation de mémoire MemExplorer-Dynamic, est que le temps d'exécution est divisé en $T$ intervalles de temps dont les durées peuvent être exceptionnelles, ces périodes sont supposées être fournies avec l'application. Pendant chaque intervalle de temps, l'application doit occuper un sous-ensemble donné de ses structures de données en lecture et/ou en écriture. Le processeur accède à la structure de données pour exécuter les commandes de l'application. Comme dans MemExplorer, lorsque l'intervalle de temps $T0$, toutes les structures de données sont en mémoire externe et les bancs de mémoire sont vides. Le temps d'entrée d'une structure de données est son nombre d'accès multiplié par le taux de transfert du processeur vers les bancs de mémoire ou la mémoire externe. Le temps de transfert du processeur vers la mémoire externe est de $p$ ms et le taux de

transfert du processeur vers une banc de mémoire est de $l$ ms. Le temps nécessaire pour déplacer une structure de données de la mémoire externe vers une banc de mémoire (et vice versa) est équivalent à la taille de la structure de données multipliée par le taux de transfert v/ms par kilooctet (ms/kB). Le temps nécessaire pour transférer une structure de données d'une banc de mémoire à une autre est égal à la taille de la structure de données multipliée par le temps de transfert entre les bancs de mémoire, c'est-à-dire $l$ ms/kB Soto *et al.* [200].

Le processeur peut avoir accès à tous les bancs de mémoire simultanément. Pour un conflit entre deux structures de données, il a été défini dans les versions précédentes. De plus, les deux cas distincts, les conflits automatiques et les structures de données isolées sont analysés dans cette version de MemExplorer-Dynamic.

La formulation de MemExplorer-Dynamic est la suivante : allouer un banc de mémoire ou une mémoire externe à toute forme de structure de données du software pour tout intervalle de temps, afin de minimiser le temps passé à accéder et à modifier la structure de données tout en satisfaisant la capacité du banc de mémoire.

### 4.5.2 Formulation mathématique

Le nombre de bancs de mémoire est indiqué par $m$, et le banc de mémoire $m+1$ fait référence à la mémoire externe. La capacité de la banc de mémoire $j$ est $c_j$ pour tous $j \in \{1, \cdots m\}$. Soit $n$ est le nombre de structures de données. La taille d'une structure de données est indiquée par $s_i$, pour tous $i \in \{1, \cdots n\}$. $n_t$ est le nombre de structures de données accessibles par l'application pendant l'intervalle de temps $I_t$, pour tous $t \in \{1, \cdots T\}$. $A_t \subset \{1, ..., n\}$ est indiqué l'ensemble des structures de données requises dans l'intervalle de temps $I_t$ pour tous $t \in \{1, \cdots T\}$.

Ainsi $e_{i,t}$ indique le nombre de fois que $i \in A_t$ est accessible dans l'intervalle $I_t$. Le nombre de conflits dans $I_t$ est désigné par $o_t$, et $d_{\{}k, t\}$ est le coût du conflit $(k, t) = (k_1, k_2)$ pendant l'intervalle de temps $I_t$ pour tous $k \in \{1, \cdots o_t\}$, $k_1$ et $k_2 \in A_t$, et $t \in \{1, \cdots T\}$. L'attribution des structures de données aux bancs de mémoire (et à la mémoire externe) pour chaque intervalle de temps est modélisée sous forme de matrice $X$, pour tous $(i, j, t) \in \{1, \cdots n\} * \{1, \cdots m+1\} * \{1, \cdots T\}$.

Les statuts des conflits sont représentés comme suit : $y_{k,t}$ pour tous $k \in 1, \cdots o_t$, et $t \subset \{1, \cdots T\}$.

Enfin, la formulation PLNE de MemExplorer-Dynamic est [200] :

$$f = \sum_{t=1}^{T} \left[ (p-1) \sum_{i \in A_t} (e_{i,t}.x_{i,m+1,t}) - \sum_{k=1}^{o_t} y_{k,t} d_{k,t} + \sum_{i \in A_t} s_i.(l.w_{i,t} + v.w\prime_{i,t}) \right] \tag{4.18}$$

$$Minimize f \tag{4.19}$$

$$\sum_{j=1}^{m+1} x_{i,j,t} = 1 \quad \forall i \in \{1, ..n\}, \forall t \in \{1, \ldots, T\} \tag{4.20}$$

$$\sum_{i \in A_t}^{n} x_{i,j,t}.s_i \leq c_j \forall j \in \{1, \ldots, m\}, \forall t \in \{1, \ldots, T\} \tag{4.21}$$

$$x_{k_1,j,t} + x_{k_2,j,t} \leq 2 - y_{k,t} \forall k_1, k_2 \in A_t, \forall j \in \{1, \ldots, m+1\}, \quad \forall\ k \in \{1, \ldots, o_t\}, \forall t \in \{1, \ldots, T\} \tag{4.22}$$

$$x_{i,j,t-1} + x_{i,g,t} \leq 1 + w_{i,t}, \quad \forall i \in \{1, \ldots, n\}, \forall j \neq g, \quad (j, g) \in \{1, \ldots, m\}^2, \forall t \in \{1, \ldots, T\} \tag{4.23}$$

$$x_{i,m+1,t-1} + x_{i,j,t} \leq 1 + w\prime_{i,t} \ , \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}, \quad \forall t \in \{1, \ldots, T\} \tag{4.24}$$

$$x_{i,j,t-1} + x_{i,m+1,t} \leq 1 + w\prime_{i,t} \forall i \in \{1, \ldots, n\}, \quad \forall j \in \{1, \ldots, m\}, \forall t \in \{1, \ldots, T\} \tag{4.25}$$

$$x_{i,j,o} = o \quad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \tag{4.26}$$

$$x_{i,m+1,o} = 1 \quad \forall i \in \{1, \ldots, n\} \tag{4.27}$$

$$x_{i,j,t} \in \{0, 1\} \quad \forall \quad i \in \{1, \ldots, n\} \forall j \in \{1, \ldots, m\}, \forall t \in \{1, \ldots, T\} \tag{4.28}$$

$$w_{i,t} \in \{0, 1\} \quad \forall \quad i \in \{1, \ldots, n\}, \forall t \in \{1, \ldots, T\} \tag{4.29}$$

$$w\prime_{i,t} \in \{0, 1\} \quad \forall \quad i \in \{1, \ldots, n\}, \forall t \in \{1, \ldots, T\} \tag{4.30}$$

$$y_{k,t} \in \{0, 1\} \quad \forall \quad k \in \{1, \ldots, o_t\}, \forall t \in \{1, \ldots, T\} \tag{4.31}$$

La fonction objectif (4.18) comporte trois termes, le premier terme est le coût d'accès de toutes les structures de données qui se trouvent dans une banc, le deuxième terme

est le coût d'accès à toutes les structures de données de la mémoire externe et le dernier est le coût du conflit fermé.

### 4.5.3 Méthodes de résolution

une méthode de résolution exacte a été proposée. La solution optimale peut être obtenue en utilisant une solution, telle que GLPK ou Xpress-MP. Pourtant, comme le montrent les tests numériques, la solution optimale ne peut être obtenue en un temps raisonnable pour les cas intermédiaires. C'est la deuxième raison des métaheuristiques suggérées. Deux métaheuristiques itératives (approche à court terme et approche à long terme), basée sur le problème général d'allocation de mémoire. Les résultats numériques montrent que le processus à long terme donne de bons résultats en un temps raisonnable, ce qui rend cette approche appropriée pour les besoins d'aujourd'hui et de demain. Le principal inconvénient de cette approche est qu'elle ignore la possibilité de mettre à jour la solution à chaque itération [200].

Il y a d'autres améliorations au modèle PLNE fourni dans le Memexplore, où il a été amélioré par une étude qui a attaqué le MemExplorer-Dynamic, et une nouvelle approche métaheuristique est proposée, basée sur le VNS (voir [108]).

## 4.6 Gestion de la mémoire Scratch-Pad

Ce problème est lié au problème de l'optimisation du logiciel qui a été expliqué dans le chapitre 3 (sous section 3.4). Il est contraire aux versions présentées précédemment dans Soto *et al.* [200], qui lient par liaison de données.

Une allocation de manière optimale le code et/ou les données de l'application à une mémoire typique appelée mémoire Scratch-Pad afin de diminuer la consommation d'énergie des systèmes embarqués Idrissi Aouad *et al.* [105].

### 4.6.1 Description du problème

La réduction de la consommation d'énergie des systèmes embarqués nécessite une gestion particulière de la mémoire. Il a été confirmé que les mémoires de type "Scratch Pad"

(SPM) sont des structures de données de faible capacité, peu coûteuses et adaptées (c'est-à-dire économes en énergie) qui sont immédiatement contrôlées et gérées au niveau logiciel Idrissi Aouad *et al.* [105].

La mémoire de type Scratch Pad (SPM) est une mémoire interne très rapide utilisée pour le stockage temporaire de calculs, de données et d'autres travaux d'amélioration. La SPM est traitée immédiatement et explicitement au niveau du logiciel, soit par le développeur, soit par le compilateur.

Le problème qui se pose ici ressemble à un problème de sac à dos. En effet, en raison de la petite taille des SPMs, l'allocation des données doit être optimale afin de réduire la consommation d'énergie. Par exemple, la majorité des auteurs utilisent l'une de ces trois règles : D'abord, ils allouent les données aux SPMs en fonction de leur taille. Ensuite, ils allouent les données aux SPMs en fonction du nombre d'accès et enfin, ils autorisent le stockage des données dans les SPMs en fonction du nombre d'accès et de la taille (BEH) Idrissi Aouad *et al.* [105].

Une architecture mémoire composée d'une mémoire Scratch-Pad, une mémoire principale (DRAM) et d'une mémoire cache d'instructions.

L'optimisation s'étend à un nombre variable de bancs de mémoire [200]. A l'inverse, Shyam and Govindarajan [196] ils s'attachent à minimiser la consommation d'énergie dans un tel système en fixant l'allocation des bancs de manière que certains d'entre eux puissent être mis en attente le plus possible. Tous ces travaux analysent des systèmes monoprocesseurs et s'appuient sur des techniques d'analyse de conflits qui ne sont pas applicables aux systèmes multiprocesseurs. Voir aussi ([114]), c'est une étude concerne le domaine des systèmes multicœurs.

Nous trouvons de nombreux travaux ont porté sur le problème de l'allocation de mémoire dans les systèmes embarqués et ont proposé des modèles utilisant la programmation linéaire pour réduire la consommation d'énergie. Ces travaux de recherche sont les suivants : [14, 16, 20, 48, 57, 103, 104, 118, 135, 185, 192, 193, 198, 204, 220, 221].

### 4.6.2 Formulation mathématique

Un modèle de mesure de la consommation énergétique de la mémoire a été proposé, il étudie une architecture composée d'un SPM, de DRAM et d'une instruction de cache.

Dans ce modèle, il existe deux stratégies de cache : write-through et write-back. Dans un cache Write-Through ($WT$), chaque écriture dans le cache entraîne une écriture synchrone dans la DRAM. Dans un cache Write-back ($WB$), les écritures ne sont pas directement répercutées dans la mémoire principale. Au lieu de cela, le cache suit les emplacements qui ont été écrits et marque ces emplacements comme étant sales. Les données contenues dans ces emplacements sont réécrites dans la DRAM lorsque ces données sont vidées du cache.

le modèle mathématique est [105] :

liste des termes :

$E_{tspm}$ :Énergie totale consommée dans SPM.
$E_{tic}$ : Énergie totale consommée dans le cache d'instructions.
$E_{tdram}$ : Énergie totale consommée en DRAM.
$E_{spmr}$ : Énergie consommée lors d'une lecture de SPM.
$E_{spmw}$ :Énergie consommée lors d'une écriture dans SPM.
$N_{spmr}$ :Lecture du numéro d'accès à SPM.
$N_{spmw}$ :Écriture du numéro d'accès à SPM.
$E_{icr}$ : Énergie consommée lors d'une lecture à partir du cache d'instructions.
$E_{icw}$ :Énergie consommée lors d'une écriture dans le cache des instructions.
$N_{icr}$ :Lecture du numéro d'accès au cache d'instructions.
$N_{icw}$ :Écriture du numéro d'accès au cache d'instructions.
$E_{dramr}$ :Énergie consommée lors d'une lecture de DRAM.
$E_{dramw}$ :Énergie consommée lors d'une écriture dans DRAM.
$N_{dramr}$ :Lecture du numéro d'accès à DRAM.
$N_{dramw}$ :Écriture du numéro d'accès à DRAM.

$WP_i$ la politique d'écriture de cache considérée : $WT$ ou$WB$. Dans le cas de $WT, WP_i = 1$ dans le cas de $WB$ alors $WP_i = 0$.

$DB_{i_k}$ Dirty Bit utilise dans le cas de $WB$ pour indiquer lors de l'accès $k$ si la ligne de cache d'instructions a été modifiée avant ($DB_i = 1$) ou non ($DB_i = 0$).

$h_{i_k}$ type de l'accès $k$ au cache d'instructions. En cas de succès du cache, $h_{i_k} = 1$. En cas de cache manqué, $h_{i_k} = 0$.

$$E = E_{tspm} + E_{tic} + E_{tdram}$$

$$E = N_{spmr} \star E_{spmr} + N_{spmw} \star E_{spmw} \tag{4.32}$$

$$+ \sum_{k=1}^{N_{icr}} [h_{i_k} \star E_{icr} + (1 - h_{i_k}) \star [E_{dramr} + E_{icw} + (1 - WP_i) \star DB_{i_k} \star (E_icr + E_dramw)]] \tag{4.33}$$

$$+ \sum_{k=1}^{N_{icr}} [WP_i \star E_dramw + h_{i_k} \star E_icw + (1 - WP_i) \star (1 - h_{i_k}) \star [E_icr + DB_{i_k} \star (E_icr + E_dramw)]] \tag{4.34}$$

$$+ N_dramr \star E_dramr \tag{4.35}$$

$$+ N_dramw \star E_dramw \tag{4.36}$$

L'équation (4.32) représente respectivement l'énergie totale consommée lors d'une lecture et lors d'une écriture de/en SPM. Les équations (4.33 et 4.34) représentent respectivement l'énergie totale consommée lors d'une lecture et lors d'une écriture de/en cache d'instructions. Quand, Les équations (4.35 et 4.36) représentent respectivement l'énergie totale consommée lors d'une lecture et pendant une écriture de/vers DRAM.

### 4.6.3 Méthodes de résolution

Développement d'une solution semi-automatique qui est basée sur le solveur MATLAB et sur l'outil d'instrumentation de code Gcov (Source Code Coverage). Une autre solution proposée est une métaheuristique tabou de recherche (TS) pour la gestion de l'allocation de mémoire qui est une nouvelle alternative originale à la méthode existante la plus connue (BEH) [105, 106].

Il existe plusieurs travaux basés sur le modèle d'estimation de la consommation d'énergie proposé ici (voir [104]), et résolus par des heuristiques évolutionnaires (algorithmes génétiques, processus de décision de Markov, recuit simulé, méthode ANT, technique de l'essaim de particules.

## 4.7 Conclusion

Dans les systèmes embarqués, le coût de la consommation en énergie est l'un des problèmes les plus sensibles, en raison de la complexité du trafic d'informations et des fonctions multimédia. Ce problème est résolu à l'aide de différents algorithmes et utilisé dans la littérature précédente, mais ne fournit pas la meilleure solution optimale. Il y a beaucoup de recherches pour essayer de trouver la meilleure façon de résoudre ce problème, jusqu'à ce que les résultats obtenus soient assez encourageants en ce qui concerne la consommation d'énergie de la mémoire. Ces résultats ont mis en évidence l'importance de réduire la consommation d'énergie.

Le système de mémoire reste l'un des problèmes de recherche actuels dans la conception des systèmes embarqués. Le système mémoire restera également la principale source de performance et de consommation d'énergie des systèmes embarqués.

Le chapitre (3) donne un aperçu des techniques d'optimisation de la mémoire pour les systèmes embarqués. Nous avons passé en revue quelques techniques d'optimisation de la mémoire, ces techniques étant axées soit sur l'optimisation matérielle, soit sur l'optimisation logicielle, l'optimisation des liaisons de données (optimisation matérielle / logicielle). Également dans ce chapitre actuel (4), nous avons expliqué le problème de l'allocation de mémoire en présentant les solutions proposées dans la littérature qui s'intéresse à ce domaine.

Le tableau (4.1) ci-dessous fournit un résumé général de toutes les nouvelles versions du problème d'allocation de mémoire dans les systèmes embarqués, ainsi que les méthodes de solution que les chercheurs ont présentées dans leurs études.

Tous les problèmes d'allocation de mémoire sont des problèmes NP-difficiles. Pour chaque problème, le nombre d'entrées augmente, la fonction objectif change, et les caractéristiques du problème d'allocation de mémoire sont modifiées. Ainsi, chaque problème possède des modèles mathématiques précis et une approche exploratoire. Les chercheurs doivent démontrer que les résultats produits par les métaheuristiques proposées sont meilleurs en termes de fonction objectif et de temps d'exécution que ceux obtenus par les solutions PLNE et de recherche.

| Version du problème MAP | Le problème est lié à | Les techniques d'optimisation | Méthodes de résolution exacte | Meatheuristique |
|---|---|---|---|---|
| Sans contrainte [200] | Optimisation du matériel | Proposer le PLNE | GLPK ou Xpress | Proposer des bornes supérieures sur le nombre chromatique |
| Contrainte sur le nombre de bancs de mémoire [200] | Liaison des données (codesign matériel/logiciel) | Proposer le PLNE | GLPK ou Xpress | *Evo-Allocation basée sur un algorithme évolutionnaire hybride<br>*Allocation taboue basée sur la méthode de recherche taboue |
| Général [200] | Liaison des données (codesign matériel/logiciel) | Proposer le PLNE | GLPK ou Xpress | *VNS-TS-Memexplorer basé sur la recherche tabou et la recherche locale |
| Dynamique [108, 192, 200] | Liaison des données (codesign matériel/logiciel) | Proposer le PLNE | GLPK ou Xpress | *Approche à court terme et approche à long terme basé sur une recherche locale<br>*GRASP Heuristique basée sur des procédures de construction greedy randomisées et une procédure de recherche locale<br>*Recherche de voisinage à deux variables algorithmes : (Base VNS) et (RSVNS) |
| Gestion de la mémoire du Scratch-Pad [105, 106] | Optimisation des logiciels | Proposer le PLNE | Solution semi-automatique basée sur Matlab | *Heuristique de recherche Tabu (TS)<br>*Heuristique (BEH) |

TABLE 4.1 – Résumé sur les versions des problèmes d'allocation de mémoire

# Chapitre 5

# Combinaison de la réduction des données, du solveur MIP et de la recherche locale itérée pour l'affectation généralisée

## 5.1 Introduction

Trouver une affectation de tâches aux machines au coût minimal, de telle sorte que chaque tâche soit affectée à un seul machine et que la capacité des ressources de chaque machine soit respectée, est le problème d'affectation généralisé de base (GAP). L'ordonnancement des tâches, le routage, le chargement des systèmes de fabrication flexibles et la localisation des installations sont des applications de ce problème NP-difficile. L'objectif de ce chapitre est de présenter le problème d'affectation généralisé (GAP), certaines de ses applications pratiques et de ses propriétés (Section 5.1.1), de motiver l'approche que nous proposons (Section 5.1.2), et de résumer nos résultats (Section 5.1.3).

### 5.1.1 Problème d'affectation généralisé (GAP)

GAP, ses différentes contraintes telles que le sac à dos, le sac à dos multiple, le bin packing, et ses extensions telles que le GAP goulot, multiniveau, stochastique et quadratique, sont des problèmes d'optimisation combinatoire fondamentaux qui ont attiré beaucoup d'attention. GAP a servi de banc d'essai pour la plupart des paradigmes de la recherche opérationnelle. Une étude approfondie qui couvre presque tous ses aspects peut être trouvée dans [154]. Voir aussi la vaste revue de la littérature dans [92].

Tout d'abord, présentons un modèle mathématique de GAP. Supposons que nous devions effectuer $n$ tâches sur $m$ machines. Soit $b_i$ la disponibilité des ressources de la machine $i$, soit $a_{ij}$ la quantité de ressources requise par la machine $i$ pour effectuer le tâche $j$ et soit $c_{ij}$ le coût d'affectation de la tâche $j$ à la machine $i$. Chaque tâche est contrainte d'être assignée à une seule machine sans dépasser la disponibilité de ses ressources. GAP consiste à trouver une affectation à coût minimal sous la contrainte ci-dessus. Un modèle IP bien connu pour GAP est :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, \cdots, n \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \qquad i = 1, \cdots, m \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \cdots, m, j = 1, \cdots, n \tag{4}$$

où $x_{ij}$ indique si la tâche $j$ est affecté ou non à la machine $i$.

Une affectation est un $m \times n$-vector $x$ satisfaisant (2-4). Au lieu du vecteur $x$, nous préférons utiliser $n$ -vector $s$ avec $s_j \in \{1, \cdots, m\}$ pour tout $j, j = 1, \cdots, n$ pour que $s_j = i \Leftrightarrow x_{ij} = 1$.

Du point de vue de la complexité informatique, GAP-faisabilité, le problème de décision qui demande une affectation faisable, est NP-complet. Ce fait implique la forte NP-diversité de GAP. De plus, GAP est APX-dur [38].

Depuis sa création, GAP a connu un grand nombre d'applications de premier plan et la liste continue de s'allonger. Dans leur étude Barbas and Marín [17] ont proposé une heuristique de décomposition lagrangienne pour un problème de conception de réseau. Un modèle GAP non linéaire a été présenté dans [31] pour l'affectation de travailleurs polyvalents. Modélisation et résolution du problème de déneigement de la ville de Montréal sous forme de GAP [32]. Résoudre un problème d'affectation des terres en assignant de manière optimale les parcelles de terrain aux activités d'utilisation des terres. [49]. Aussi dans ce travail Dobson and Nambimadom [60], une formulation de programmation en nombres entiers et une heuristique GAP pour le problème du chargement et de l'ordonnancement des lots. Une variation de GAP dans l'industrie laitière en néo-zélandaise étudiée dans [72]. Un problème lié à l'approvisionnement en canne à sucre dans une région de sucreries en Australie est abordé en Higgins [100]. La répartition des objets observés par le télescope ROSAT à des intervalles de temps a été modélisée et résolue comme GAP avec des contraintes secondaires par Nowakovski *et al.* [152]. L'affectation des patients militaires et de leurs membres de leur famille à des séquences d'aéronefs dans une flotte aéromédicale de l'US Air Force Ruland [181].

En raison de sa pertinence pratique, GAP a été largement étudié d'un point de vue algorithmique, que ce soit de manière exacte ou approximative. Parmi les méthodes exactes, nous trouvons la méthode de branch and bound [13], et beaucoup Branch-and-Bound [36, 87, 147, 171], et Branch-and-Price [79, 187], et Branch-and-Cut-and-Price [168], et Column Generation (Sadykov *et al.* [182]) sont parmi les autres méthodes qui ont été suggérées.

Mais lorsque le temps est plus important que la qualité de la solution, les métaheuristiques sont recommandées, car elles trouvent des solutions quasi optimales dans des temps de calcul tolérables. Ces méthodes comprennent les algorithmes génétiques [44, 128, 224], Recherche Tabu [59], Recherche tabou avec chaîne d'éjection [226], Recherche VLSN [80, 92, 138], Réseau neuronal [140], Colonie d'abeilles [155], Renvoi de chemin avec chaîne d'éjection [228], Differential evolution [191, 202], et l'hybridation [94].

### 5.1.2 Motivation

L'allocation de mémoire est un défi majeur pour la conception électronique des systèmes embarqués, car elle affecte directement et fortement leur consommation d'énergie et leurs performances. En effet, l'allocation optimale des structures de données dans les bancs de mémoire permet généralement de réaliser des bénéfices substantiels en termes de consommation d'énergie, et donc d'utilisation des batteries à long terme.

Considérons une MAP spécifique et sa formulation mathématique empruntée à [201] et [199]. Le système embarqué considéré comprend $m$ bancs de mémoire, chacune avec une capacité limitée $c_i, i = 1, \cdots, m$ et une mémoire externe avec une capacité illimitée supposée (*i.e.* $c_{m+1} = \infty$). D'autre part, un code informatique (par exemple un `C` source code) impliquant des structures de données $n$ est exécuté sur le système. La structure de données $j$ a une taille $s_j$ (généralement en kilo-octets). L'accéder à la structure de données $j$ à partir de n'importe quelle banc de mémoire prend du temps $t_j, j = 1, \cdots, n$ alors qu'il en faut beaucoup plus pour y accéder depuis la mémoire externe. Le temps d'accès de la structure de données $j$ à partir de la mémoire externe est de $\theta \times t_j$ où $\theta$ est un nombre donné.

Il y a $p$ paires de structures de données conflictuelles $(k_1, k_2), k = 1, \cdots, p$. Deux structures de données sont dites conflictuelles si elles doivent être accédées simultanément. De

plus, leur conflit est dit fermé si elles sont alloués à la même banc mémoire, auquel cas le conflit génère un temps d'accès supplémentaire $d_k$ puisqu'elles doivent être accédées séquentiellement. Le conflit entre deux structures de données est ouvert si elles sont allouées à deux bancs de mémoire différents. Dans ce cas, le temps d'accès supplémentaire est nul.

Introduisons des variables binaires :

$$x_{ij} = \begin{cases} 1 & \text{si la structure de données } i \text{ est allouée à la banc de mémoire } j \\ 0 & \text{autrement} \end{cases}$$

et

$$y_k = \begin{cases} 1 & \text{si le conflit } k \text{ est fermé} \\ 0 & \text{autrement} \end{cases}$$

Le modèle mathématique est le suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} t_j x_{ij} + \theta \sum_{j=1}^{n} t_j x_{m+1,j} - \sum_{k=1}^{p} d_k y_k \tag{5}$$

$$\sum_{i=1}^{m+1} x_{ij} = 1 \qquad j = 1, \cdots, n \tag{6}$$

$$\sum_{j=1}^{n} s_j x_{ij} \leq c_i \qquad i = 1, \cdots, m \tag{7}$$

$$x_{k_1,j} + x_{k_2,j} \leq 2 - y_k \qquad i = 1, \cdots, m+1 \text{ and } k = 1, \cdots, p \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \cdots, m \text{ and } j = 1, \cdots, n \tag{9}$$

$$y_k \in \{0, 1\} \qquad k = 1, \cdots, p \tag{10}$$

La fonction de coût dans (5) est le temps total (exprimé en millisecondes) nécessaire pour accéder à toutes les structures de données. La contrainte (6) exprime que chaque structure de données est affectée à une seule banc de mémoire (qui peut être la mémoire externe). La contrainte (7) détermine que la taille totale des structures de données affectées à la banc mémoire $i$ ne doit pas dépasser sa capacité. La contrainte de type (8) traite les conflits entre structures de données. Si $y_k = 0$ alors la contrainte est redondante,

car le conflit est ouvert tandis que $y_k = 1$ interdit d'allouer les deux structures de données à la même banc de mémoire. Enfin, toutes les variables sont binaires.

Si nous omettons la contrainte (8) liées aux conflits avec la contrainte (10) et le terme $-\sum_{k=1}^{p} d_k y_k$ dans la fonction objectif, le modèle devient un GAP. La conception d'une méthode de solution rapide et suffisamment précise pour cette dernière est donc notre principale motivation.

Dans une étude future, cet algorithme sera utilisé pour obtenir une bonne affectation (au sens de la valeur de la fonction objectif (5)) satisfaisant les contraintes (6, 7, 9). Mais cette solution ne satisfait guère les contraintes de conflit (8). L'étape suivante consiste à le réparer en définissant une mesure de faisabilité et en utilisant une méta-heuristique de recherche locale pour 'améliorer' la solution construite jusqu'à ce qu'elle devienne faisable. La dernière étape consiste à utiliser, encore une fois, la recherche locale pour améliorer la solution du point de vue de la fonction objectif (5).

Au-delà de la vaste littérature sur le MAP, il existe des articles scientifiques récemment publiés ([223, 231, 233, 234]) qui traitent de problèmes similaires, dans le sens où les objets doivent être affectés de manière optimale à (ou appariés avec) d'autres objets tout en évitant les conflits. Par conséquent, les méthodes proposées dans ces articles pour des problèmes similaires peuvent aider à la conception de méthodes pour le MAP. Cette direction mérite d'être explorée.

### 5.1.3 Résumé de la méthode suivie pour résoudre GAP

L'importance pratique de GAP et sa difficulté de calcul rendent toute contribution à la conception d'une nouvelle méthode de calcul rapide et/ou efficace très importante. à la conception d'une nouvelle méthode de calcul rapide et/ou efficace. La méthode que nous que nous proposons dans ce travail commence par une heuristique de réduction des données dont le but est de réduire la taille de GAP. Cette réduction de données est maintenant devenue standard et bien établie puisque puisqu'elle a été largement utilisée dans de nombreux contextes : GAP [93], Couverture de l'ensemble [89] et Liste des infirmières [85]. En fait, elle est générique et peut traiter tout problème d'optimisation combinatoire, à condition que ses variables soient binaires. Dans pratique, la réduction des données permet d'éliminer jusqu'à 96% des variables de GAP. La version réduite

de GAP réduit laissé par cette procédure est petit et clairsemé. En outre, sa solution optimale peut être facilement être étendue à une solution optimale ou quasi-optimale de GAP.

Nous verrons plus loin que le jeu de données de benchmark utilisé dans cette étude est constitué d'instances difficiles. Une façon naturelle de les traiter est de soumettre l'instance GAP réduite, obtenue en tant que résultat de la procédure de réduction des données, à un solveur MIP. En effet, les résultats obtenus par le solveur `Cplex 12.6` sont satisfaisants du point de vue de la précision, ainsi que du temps de calcul.

Cependant, nous pouvons faire mieux en utilisant ILS pour résoudre le GAP réduit. Cette métaheuristique est une heuristique de haut niveau conçue pour piloter une heuristique LS de bas niveau afin de trouver de bonnes solutions à un problème d'optimisation en échantillonnant l'ensemble de solutions qui est trop grand pour être recherché complètement. L'efficacité de l'ILS est confirmée par un nombre incroyable d'articles publiés au cours de la dernière décennie environ. Voir par exemple l'article récemment publié ([96]) qui hybride l'ILS et le recuit simulé. Le schéma de base de l'ILS peut être résumé comme suit : l'optimum local actuel $\sigma$ est perturbé pour atteindre une solution intermédiaire $s$ ; LS est ensuite appliqué à $s$ et obtient un nouvel optimum local $\sigma'$. Si ce dernier passe le test d'acceptation, il devient la prochaine solution à perturber ; sinon, on revient à l'optimum local précédent $\sigma$. Ces étapes sont répétées jusqu'à ce qu'un critère d'arrêt soit atteint.

Les principaux éléments constitutifs de l'ILS sont la solution initiale, le LS intégré et la phase de perturbation. Dans notre proposition, la solution initiale est obtenue en appliquant le solveur Cplex (conçu comme une heuristique) pour une courte période de temps. Les voisinages considérés sont deux ensembles simples de taille $O(mn)$ et $O(n^2)$ . Le mécanisme de perturbation consiste à modifier légèrement la matrice des coûts, puis à appliquer la recherche locale pour obtenir un sortant qui sera amélioré par la recherche locale en utilisant la matrice de coût d'origine.

Notre méthode, combinant la réduction des données et l'ILS, est testé sur un benchmark largement utilisé d'instances larges et difficiles. Les expériences menées montrent que cette méta-heuristique fournit des solutions GAP de bonne qualité en un temps très court, et c'est exactement ce qu'on attend d'elle. L'organigramme de la méthode proposée, qui constitue un résumé graphique, est donné dans la figure 5.1.

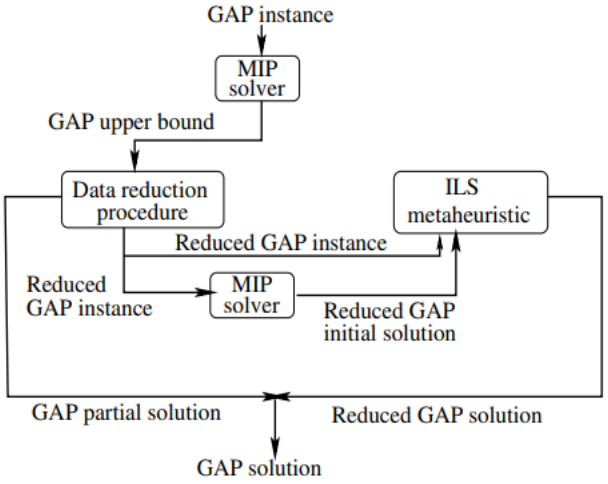


FIGURE 5.1 – L'organigramme de la méthode proposée

Le chapitre est organisé comme suit. La section 5.2 récapitule l'heuristique générique de réduction des données. La section 5.3 détaille les étapes de l'ILS. La section 5.4 est consacrée aux expériences de calcul et à la comparaison de notre méthode avec trois autres méthodes publiées récemment.

## 5.2 Heuristique de réduction des données : un rappel

Comme déjà remarqué, cette section rappelle l'heuristique de réduction des données qui a été détaillée dans [93]. Cependant, comme cette méthode est une partie essentielle de notre approche, elle mérite d'être décrite en détail. Appelons RGAP le GAP réduit résultant de l'application de l'heuristique de réduction des données.

Considérons la relaxation lagrangienne des contraintes de sac à dos (3) de GAP. Si nous prenons $\pi \in \mathbb{R}^m_-$ être un vecteur multiplicateur lagrangien non positif associé aux contraintes de sac à dos relâchées, le dual lagrangien de GAP est le problème $\max_{\pi \in \mathbb{R}^m_-} w(\pi)$ où $w(\pi)$ est la valeur optimale du problème

$$L(\pi) \begin{cases} w(\pi) = & \sum_{i=1}^{m} \pi_i b_i + \min \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} - \pi_i a_{ij}) x_{ij} \\ & \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1, \cdots, n \\ & x_{ij} \in \{0,1\} \quad i = 1, \cdots, m, j = 1, \cdots, n \end{cases}$$

Résoudre exactement le dual lagrangien pour une valeur optimale $\pi^*$ est très difficile parce que $w(\pi)$ est une fonction linéaire non différentiable concave par morceaux. Heureusement, l'optimisation des sous-gradients peut être utilisée pour trouver une solution quasi optimale $\pi'$. La méthode du sous-gradients commence par $\pi^{(0)} = 0$ et génère une séquence de vecteurs $\pi^{(t)}, t \geq 1$. À la fin de la méthode, $\pi' = \text{argmax}_{t \geq 0} w(\pi^{(t)})$ est considérée comme une bonne approximation.

Dans l'itération $t$ avec le vecteur $\pi^{(t)}$ fixé, on appelle $x^{(t)}$ la solution du problème $L(\pi^{(t)})$ qui est facilement calculé(en temps $O(mn)$). Il suffit, pour chacun $j, j = 1, \cdots, n$, trouver :

$$i' = \text{argmin}_i(c_{ij} - \pi_i a_{ij}) \tag{11}$$

et mettre en place

$$x_{i'j}^{(t)} = 1 \text{ and } x_{ij}^{(t)} = 0 \text{ for } i = 1, \cdots, m, i \neq i' \tag{12}$$

Une fois la solution $x^{(t)}$ du problème relâché est connu, nous pouvons calculer le sous-gradient $m$-vector $v^{(t)}$ dont $i^{\text{th}}$ le composant est :

$$v_i^{(t)} = b_i - \sum_{j=1}^{j=n} a_{ij} x_{ij}^{(t)} \tag{13}$$

Depuis la solution $x^{(t)}$ et le sous-gradient $v^{(t)}$ sont connus, nous pouvons mettre à jour chaque composant du lagrangien $m$-vector $\pi^{(t)}$ comme suit :

$$\pi_i^{(t+1)} \leftarrow \max \left\{ 0, \pi_i^{(t)} + \rho \frac{ub - w(\pi^{(t)})}{\|v^{(t)}\|^2} \right\} \tag{14}$$

Le nombre $\rho$ est le coefficient de relaxation. Une bonne règle empirique est de commencer avec une valeur de 2, puis de la diviser par deux à chaque nombre fixe d'itérations.

Soit $N_{DR}$ le nombre d'itérations de la méthode du sous-gradient. Dès que la méthode des sous-gradients s'arrête, nous recherchons les variables qui ne sont jamais, ou seulement rarement sélectionnées dans la solution du problème relaxé. Pour cette raison, nous calculons :

$$f_{ij} = \frac{1}{N_{DR}} \sum_{t=0}^{N_{DR}-1} x_{ij}^{(t)} \text{ for } i = 1, \cdots, m \text{ and } j = 1, \cdots, n \tag{15}$$

où $0 \leq f_{ij} \leq 1$.

La valeur $f_{ij}$ est la fréquence de l'implication de la variable $x_{ij}^{(t)}$ de valeur 1 dans le problème relaxé. Soit $\Phi$ un petit nombre positif proche de 1. Par conséquent, notre idée intuitive est d'éliminer chaque variable $x_{ij}$ du problème chaque fois que la valeur correspondante de $f_{ij}$ est inférieure au filtre donné $\Phi$ (*i.e.* $f_{ij} \leq \Phi \Rightarrow x_{ij} = 0$). En effet, l'affectation de la tâche $j$ à la machine $i$ dans une solution optimale de GAP est très peu probable lorsque $f_{ij}$ est proche de zéro.

Pour chaque tâche $j$, soit $M_j = \{i \in \{1, \cdots, n\}, f_{ij} > \Phi\}$ être l'ensemble des machines capables d'exécuter la tâche $j$. Nous devons éviter les grandes valeurs de $\Phi$ pour garantir que $M_j \neq \varnothing$. Il est clair que lorsque $|M_j| = 1$ alors la tâche $j$ peut être allouée pour toujours à une seule machine $i^* \in M_j$ en laissant $s_j = i^*$. Soit $p = n - |\{j \in N, |M_j| = 1\}|$ être le nombre des tâches non assignées, soit $P = \{k_1, \cdots, k_p\}$ être l'ensemble des tâches non attribuées et soit $pcost = \sum_{\{j \in N, |M_j|=1\}} c_{s_j j}$ le coût de la solution partielle. Par conséquent $|M_j| \geq 2, j \in P$.

Pour aider le lecteur à reproduire la méthode, un pseudo-code détaillé est proposé dans l'algorithme 5.1. Les valeurs $f_{ij}$ sont initialisées à la ligne 1. Puisque nous avons besoin d'une borne supérieure *ub* sur la valeur optimale de GAP, nous calculons une affectation à la ligne 2 avec un coût *ub*. Le vecteur lagrangien et le coefficient $\rho$ sont initialisés aux lignes 3-4. Si nous excluons la ligne 7 qui met à jour les valeurs $f_{ij}$, les lignes 5 à 10 exécutent la méthode du sous-gradient. Les valeurs $f_{ij}$ sont « normalisées » à la ligne 11. Les lignes 12 à 21 implémentent la réduction des données et sort les données du RGAP (à la ligne 22) qu'ils stockent dans un fichier texte. Ce fichier sera lu par la suite par la deuxième partie du code exécutant la méthode ILS.

La méthode de réduction des données est contrôlée par trois paramètres : $N_{DR}$ le nombre d'itérations de la méthode des sous-gradients, le filtre $\Phi$ et le délai $t_{DR}$ donné à Cplex pour trouver une affectation.

Trouver une affectation pour une instance GAP dans la ligne 2 est un problème NP-complet. Dans notre implémentation, nous utilisons le solveur MIP `Cplex 12.6` pendant un temps très court $t_{DR}$ pour trouver une solution GAP. Puisque le nombre $t_{DR}$ d'itérations de la méthode des sous-gradients est une constante indépendante de la taille de l'instance, un simple comptage nous indique que, si nous omettons la ligne 2, l'algorithme est très rapide puisqu'il s'exécute en temps $O(m \times n)$.

**Algorithme 5.1** Heuristique de Réduction des Données

**Require:** GAP's data, $N_{DR}$, $\Phi$, $t_{DR}$
**Ensure:** RGAP's Data
1: $f_{ij} \leftarrow 0$ for $i = 1, \cdots, m$ and $j = 1, \cdots, n$
2: Compute an assignment for GAP instance {▷ Let $ub$ be its cost}
3: $\rho \leftarrow 2$
4: $\pi_i^{(0)} \leftarrow 0$ for $i = 1, \cdots, m$
5: **For** $t = 0, 1, \cdots, N_{DR} - 1$ **do**
6: Solve problem $L(\pi^{(t)})$ as in (11-12) {▷ Let $x_{ij}^{(t)}$ be the opt. sol. and $w(\pi^{(t)})$ its value}
7: $f_{ij} \leftarrow f_{ij} + x_{ij}^{(t)}$ for $i = 1, \cdots, m$ and $j = 1, \cdots, n$
8: Compute the subgradient vector $v^{(t)}$ as in (13)
9: Update the Lagrangian vector $\pi_i^{(t)}$ as in (14)
10: Decrease the value of $\rho$ if appropriate **EndFor**
11: $f_{ij} \leftarrow f_{ij}/N_{DR}, i \in M, j \in N$
12: $pcost \leftarrow 0$
13: $p \leftarrow n$
14: **For** $j = 1, \cdots, n$ **do**
15: $M_j \leftarrow \{1, \cdots, m\}$
16: **For** $i = 1, \cdots, m$ **do if** $f_{ij} \leq \Phi$ **then** $M_j \leftarrow M_j - \{i\}$
17: **For**$j = 1, \cdots, n$ **do if** $|M_j| = 1$ **then** {▷ Let $i^*$ be the single machine in $M_j$}
18: $p \leftarrow p - 1$
19: $s_j \leftarrow i^*$ {▷ update the partial solution}
20: $pcost \leftarrow pcost + c_{i^*j}$ {▷ and its partial cost}
21: $b_{i^*} \leftarrow b_{i^*} - a_{i^*j}$ {▷ and the resource availability of machine $i^*$}
22: **output** $p, P = \{j \in N, |M_j| \geq 2\}, (M_j)_{j \in P}, pcost, (b_i)_{i \in M}$

Le problème réduit RGAP est une restriction de GAP puisqu'il est obtenu à partir de GAP en ajoutant certaines contraintes de type $x_{ij} = 0$. Par conséquent, toute solution de RGAP peut être étendue à une solution de GAP. Les arguments empiriques [93] ont montré que RGAP capture les propriétés combinatoires de GAP, et que RGAP est une version très condensée de GAP dans le sens où il est très clairsemé et que sa solution partielle trouvée pendant la réduction des données peut être étendue à une solution optimale ou quasi-optimale de GAP. De plus, et c'est le plus important, il a été montré dans la référence ci-dessus que RGAP ne contient pas de solutions qui sont 'loin ' de l'optimal.

Malheureusement, toutes ces belles propriétés de RGAP doivent être traitées avec prudence, car l'algorithme de réduction des données proposé n'est qu'une heuristique. Si nous devions résoudre GAP exactement, l'algorithme 5.1 devrait être abandonné puisqu'une solution optimale n'est pas garantie.

Un exemple d'illustration (emprunté à [94]) impliquant des 3 machines et des 8 tâches

est donnée ci-dessous dont la solution optimale est $s = (s_1 = 3, s_2 = 3, s_3 = 1, s_4 = 1, s_5 = 2, s_6 = 2, s_7 = 1, s_8 = 2)$ du coût 104.

$$
\begin{aligned}
(c_{ij}) &= \begin{bmatrix} 15 & 30 & 30 & 26 & 18 & 11 & 1 & 29 \\ 28 & 37 & 5 & 33 & 6 & 17 & 41 & 8 \\ 8 & 8 & 22 & 33 & 23 & 23 & 39 & 8 \end{bmatrix} \\
(a_{ij}) &= \begin{bmatrix} 21 & 13 & 9 & 5 & 7 & 15 & 5 & 24 \\ 20 & 8 & 18 & 25 & 6 & 6 & 9 & 6 \\ 16 & 16 & 18 & 24 & 11 & 11 & 16 & 18 \end{bmatrix} \\
(b_i) &= (26, 25, 34)
\end{aligned}
$$

Lorsque la méthode du sous-gradient se termine, nous calculons

$$
(f_{ij}) = \begin{bmatrix} 0.110 & 0.060 & 0.330 & 0.970 & 0.113 & 0.765 & 0.993 & \boxed{0.007} \\ \boxed{0.010} & \boxed{0.025} & 0.333 & \boxed{0.007} & 0.870 & 0.203 & \boxed{0.003} & 0.815 \\ 0.880 & 0.915 & 0.338 & \boxed{0.022} & \boxed{0.018} & 0.033 & \boxed{0.005} & 0.177 \end{bmatrix}
$$

En choisissant $\Phi = 0.03$, nous pouvons voir que les variables correspondant aux valeurs encadrées sont exclues de toute considération ultérieure. En conséquence, les tâches numéro 4 et numéro 7 sont attribués pour être toujours à la machine 1 avec $pcost = 26 + 1 = 27$. Les données du RGAP avec seulement 6 de tâches restantes sur 8 et 14 de variables restantes sur 24 suivent :

$$
\begin{aligned}
(c_{ij}) &= \begin{bmatrix} 15 & 30 & 30 & 18 & 11 & \\ & & 5 & 6 & 17 & 8 \\ 8 & 8 & 22 & & 23 & 8 \end{bmatrix} \\
(a_{ij}) &= \begin{bmatrix} 21 & 13 & 9 & 7 & 15 & \\ & & 18 & 6 & 6 & 6 \\ 16 & 16 & 18 & & 11 & 18 \end{bmatrix} \\
(b_i) &= (16, 25, 34)
\end{aligned}
$$

On observe que RGAP a une solution optimale de coût 77 avec $s_1 = 3, s_2 = 3, s_3 = 1, s_5 = 2, s_6 = 2, s_8 = 2$. En se souvenant de la solution partielle $s_4 = s_7 = 1$, nous obtenons une solution optimale à GAP de coût $pcost + 77 = 27 + 77 = 104$.

## 5.3 Recherche locale itérée pour résoudre RGAP

Puisque notre objectif est la conception d'une méthode d'approximation pour GAP, il serait suffisant de considérer RGAP au lieu de traiter l'ensemble des données de GAP. Une fois qu'une solution à RGAP est obtenue, elle peut être étendue à une solution à GAP en complétant la solution partielle obtenue pendant l'étape de réduction des données.

La méthode d'approximation choisie est ILS. Ce choix est dicté par le fait que ILS est attrayante en raison de sa modularité et de sa simplicité conceptuelle. Ayant choisi ILS, nous allons nous intéresser à trois points : la solution initiale avec laquelle elle commence (Section 5.3.1), la structure de voisinage (Section 5.3.2), et le mécanisme de perturbation (Section 5.3.3). La dernière section donne une description pseudo-code de l'ILS.

### 5.3.1 Solution initiale

La première tâche dans ILS est de générer un point de départ de la recherche. Les études de recherche sur ILS concluent que la méta-heuristique est fortement affectée par la solution candidate initiale, car plus la qualité de celle-ci est bonne, meilleure est la solution finale.

La recherche d'une affectation est une tâche difficile, en particulier pour les instances GAP corrélées à grande échelle. Dans notre cas, nous obtenons une solution initiale GAP en utilisant simplement le solveur MIP `Cplex 12.6`. Puisque le solveur est conçu uniquement comme une heuristique, nous l'exécutons pendant une petite quantité de temps $t_{ILS}$. D'après la discussion de la fin de la section 5.2, le solveur génère de bonnes solutions même en si peu de temps (voir la section consacrée aux résultats expérimentaux).

### 5.3.2 Deux structures de voisinage bien connues

Le choix du voisinage à explorer pour la recherche locale est d'une importance essentielle. Dans cette étude, deux petites structures de voisinage bien connues [87, 90, 93], appelé `Swap` et `Shift`, sont considérées. Étant donné une solution GAP $s$, un mouvement est toute petite 'modification' de $s$.

Supposons que les tâches $j$ et $k$ soient affectés respectivement aux machines $s_j$ et $s_k$ dans la solution $s$. Un déplacement dans le premier voisinage consiste à choisir deux tâches $j$ et $k$ et à échanger les machines $s_j$ et $s_k$ dans la mesure où la solution résultante est faisable et rentable, ce qui nécessite que

$$b_{s_j} - a_{s_j j} + a_{s_k j} \geq 0$$
$$b_{s_k} - a_{s_k k} + a_{s_j k} \geq 0$$
$$c_{s_j j} + c_{s_k k} > c_{s_j k} + c_{s_k j}$$

Puisqu'il y a $O(n^2)$ façons de choisir deux tâches, la taille de `Swap` est $O(n^2)$ (c'est-à-dire qu'il faut un temps $O(n^2)$ pour l'explorer).

Un déplacement dans le voisinage `Shift` consiste à sélectionner une tâche $j$ et à le déplacer vers une machine $k \neq s_j$ à condition que le sortant soit réalisable et que le coût de la solution diminue. Nous demandons donc

$$b_k - a_{kj} \geq 0$$
$$c_{s_j j} > c_{kj}$$

Il existe $n$ façons de choisir une tâche et $m - 1$ façons de choisir une machine pour effectuer le tâche choisi. Par conséquent, la taille de `Shift` est $O(mn)$.

D'après leurs bornes de temps théoriques, il est très rapide d'explorer ces deux voisinages. Cependant, en général, les optima locaux obtenus en tant que titulaire sont faibles, et donc on ne peut pas attendre grand-chose de ces deux voisinages. Néanmoins (rappelons que nous résolvons RGAP), grâce aux bonnes propriétés combinatoires de ce dernier et malgré la faiblesse des deux voisinages, on obtient de bons résultats comme nous le verrons plus tard.

### 5.3.3 Perturbation de la meilleure affectation actuelle

Le but du mécanisme de perturbation est de transformer d'une manière ou d'une autre une affectation donnée en une autre affectation prometteuse, à proximité de laquelle un nouveau voisinage est construit où une prochaine solution optimale locale est recherchée.

Un facteur clé est la force de la perturbation ou le nombre de composants de la solution affectés par les changements. Avec une petite force, nous retournerons à l'optimum local précédent avec une forte probabilité, tandis qu'une grande force est similaire à un mauvais redémarrage aléatoire. Le défi consiste à trouver un bon compromis

Dans notre approche, nous suivons un schéma de perturbation quelque peu complexe utilisé dans [47]. Certains des coefficients de coût sont modifiés et la meilleure solution actuelle est utilisée comme point de départ de la recherche locale avec la nouvelle matrice de coût. Plus précisément, supposons que $s$ est la meilleure solution actuelle, alors certaines tâches sont choisies au hasard et les coûts $c_{s_j j}$ sont fixés à l'infini. Ensuite, la recherche locale est exécutée avec la nouvelle matrice de coût. Seul le plus petit voisinage `Shift` est analysé. Plus de détails seront donnés dans la section 5.3.4.

Une simple stratégie d'acceptation gloutonne est utilisée puisque la recherche locale est toujours exécutée dans le voisinage de la meilleure solution actuelle. dans le voisinage de la meilleure solution actuelle. La diversification est donc totalement négligée. La question de savoir s'il serait avantageux d'accepter également les solutions qui se dégradent (par exemple, avec une stratégie d'acceptation de recuit simulé) mérite d'être explorée.

### 5.3.4 Pseudo-code ILS

Une description de pseudo-code de l'ILS est donnée dans l'algorithme 5.2. Une solution de départ est construite dans la ligne 1. Trois paramètres contrôlent la procédure. Le nombre $N_{ILS}$ d'itérations, la limite de temps $t_{ILS}$ accordée au `Cplex` pour trouver la solution initiale et le pourcentage $\vartheta$ des tâches impliqués dans la phase de perturbation. Dans la ligne 4, nous initialisons la nouvelle matrice de coût qui est modifiée à la ligne 6. Une solution intermédiaire $s'$ est obtenue en perturbant la meilleure solution actuelle $s$ dans la ligne 7. La recherche locale est à nouveau appliquée à $s'$ pour obtenir un nouvel optimum local $s''$. Les lignes 9-11 mettent à jour la meilleure solution actuelle.

**Algorithme 5.2** ILS for solving RGAP

**Require:** RGAP's data, $N_{ILS}$, $t_{ILS}$, percentage $\vartheta$
**Ensure:** Best assignment $s^*$ and its cost
1: Use `Cplex` during $t_{ILS}$ seconds to compute an initial assignment $s^*$ {▷Let $ub$ be its cost}
2: **For** $t = 1, \cdots, N_{ILS}$ **do**
3: $s_j \leftarrow s^*_j, j \in P$
4: $d_{ij} \leftarrow c_{ij}, i = 1, \cdots, m, j = 1, \cdots, p$
5: Randomly select a percentage $\vartheta$ of jobs
6: For each job $j$ selected set $d_{s_j j} = \infty$
7: Apply local search to $s$ by using `Shift` to obtain intermediate solution $s'$
8: Apply local search to solution $s'$ to obtain solution $s''$ {▷ let $\kappa$ be its cost}
9: **If** $\kappa < ub$ **then**
10: $s^* \leftarrow s''$
11: $ub \leftarrow \kappa$
12: **output** $s^*$ and $ub$

## 5.4 Résultats expérimentaux

### 5.4.1 Preuve expérimentale

La méthode que nous proposons, appelée `ILS`, est codé dans `C` et s'exécute sur `Dell Optiplex 390` avec `Intel Core i3-2120` @ 3.3 GHz. Il est divisé en deux codes. Le premier code prend en entrée une instance GAP, exécute la procédure de réduction des données et stocke les données de RGAP dans un fichier. Le deuxième code lit le fichier et applique `ILS` à RGAP.

`ILS` est évalué sur un ensemble de données standard d'instances à grande échelle disponibles sur le site Web `www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap`. Les instances de type C et E sont considérées comme faciles et toutes les méthodes connues fonctionnent bien sur eux. Au contraire, les instances D sont très dures, car les coûts $c_{ij}$ et les coefficients de contrainte $a_{ij}$ sont fortement corrélés. Il y en a neuf en tout qui se situent entre $10 \times 400 = 4.000$ et $80 \times 1600 = 128.000$ variables. Nous allons passer à l'étude de Yagiura *et al.* [226] pour voir comment ces instances ont été générées. Quelques méthodes publiées ont obtenu de bonnes solutions avec ce type d'instance. À notre avis, une méthode ne peut être considérée comme une bonne approche de GAP que si elle agit bien sur les instances D.

Notre objectif expérimental dans cette section est de prouver la pertinence de `ILS` par des tests empiriques. Par conséquent, nous devons étudier deux problèmes : la qualité de la solution et la vitesse de calcul. Tous les temps de calcul sont exprimés en secondes.

### 5.4.2 Résultats de l'heuristique de réduction des données

La procédure de réduction des données comporte trois paramètres qui sont définis comme suit. Le nombre d'itérations $N_{DR}$ de la méthode du sous-gradient est de 400. Étant donné que l'heuristique de réduction des données à besoin d'une borne supérieure de la solution optimale, nous soumettons l'instance GAP à `Cplex 12.6` pendant un temps très court en fonction de la taille de l'instance, $t_{DR} = m \times n/4000$ seconds. Le filtre $\Phi$ utilisé pour éliminer les variables non prometteuses est fixé à 0.1.

Les deux paramètres $N_{DR}$ et $t_{DR}$ qui régissent la méthode du sous-gradients ne sont pas très importants, car des valeurs différentes peuvent être utilisées sans problème. Mais la valeur du filtre $\Phi$ est cruciale. Rappelons que $0 \leq \Phi < 1$. Des valeurs plus élevées de $\Phi$ (0.2 pour instance) donnent lieu à un plus grand nombre de variables rejetées, mais nous courons le risque de sur-réduire les données et, par conséquent, de perdre la solution optimale de GAP. Le choix de $\Phi$ doit équilibrer entre la qualité de la solution et notre besoin de réduire autant que possible le nombre de tâches restant et le nombre de variables GAP restantes.

Les résultats de l'heuristique sont fournis dans le tableau 5.1. La première colonne donne le nom de l'instance. La deuxième et la troisième colonnes donnent respectivement le temps accordé à `Cplex` et le coût de l'affectation trouvée pendant ce temps.

Les étiquettes de la quatrième à la septième colonne du tableau 5.1 sont explicites et fournissent les caractéristiques des données du RGAP. Par exemple, pour la dernière instance D801600, 122,868 variables hors de 128,000 sont éliminés (96%).

La dernière colonne du tableau 5.1 montre le temps de calcul total de l'heuristique de réduction des données. Comme on peut le voir, les temps de calcul sont très petits et sont dominés par la limite de temps accordée à `Cplex`. Ainsi, la complexité temporelle de la procédure de réduction des données est confirmée par des expériences. À partir du tableau 5.1, nous voyons que, par exemple l'instance D801600, 32 secondes sont allouées

à `Cplex` pour trouver une solution initiale alors que seulement 0,95 seconde est nécessaire pour le reste du calcul.

TABLE 5.1 – Résultats de l'heuristique de réduction des données

| | GAP solution | | RGAP's data | | | | |
|---|---|---|---|---|---|---|---|
| Instance | `Cplex` time | *ub* | Number of jobs $p$ | Number of variables | Max. nb. of machines per job | *pcost* | Comput. time |
| D10400 | 1.00 | 24988 | 269 | 612 | 5 | 9679 | 1.05 |
| D20400 | 2.00 | 24688 | 282 | 659 | 4 | 8987 | 2.11 |
| D40400 | 4.00 | 24647 | 299 | 749 | 5 | 7451 | 4.12 |
| D15900 | 3.38 | 55477 | 660 | 1525 | 4 | 17890 | 3.50 |
| D30900 | 6.75 | 54950 | 692 | 1651 | 5 | 15698 | 6.95 |
| D60900 | 13.50 | 54909 | 763 | 2206 | 5 | 9110 | 13.86 |
| D201600 | 8.00 | 97943 | 1247 | 2973 | 5 | 26004 | 8.25 |
| D401600 | 16.00 | 97365 | 1282 | 3234 | 6 | 22277 | 16.47 |
| D801600 | 32.00 | 97522 | 1491 | 5132 | 7 | 7109 | 32.95 |

### 5.4.3 Résolution de GAP et RGAP avec `Cplex`

Comme nous l'avons vu précédemment, les instances GAP sont volumineuses, 'dense' et difficiles, alors que les instances RGAP sont très clairsemées. En conséquence, il est assez naturel d'essayer de résoudre ce dernier avec `Cplex`, mais en fixant une limite de temps fixée à $m \times n/50$ ce qui est plutôt raisonnable (650 secondes en moyenne). Les résultats du calcul sont donnés dans le tableau 5.2. La deuxième colonne du tableau donne les valeurs les plus connues, le premier d'entre eux étant le seul optimal connu selon à Posta *et al.* [171]. Les valeurs de la fonction objectif obtenues par `Cplex` ne sont pas très loin des meilleurs, car ils ne s'écartent que d'environ 0.1% en moyenne.

Afin de ne pas induire le lecteur en erreur, nous rappelons que `Cplex` est appliqué à RGAP, pas à GAP et les bons résultats obtenus sont dus aux belles propriétés combinatoires de RGAP comme démontré dans [93].

À ce point de notre discussion, le lecteur peut se demander : pourquoi ne pas appliquer `Cplex` directement aux instances GAP ? faisons ces expériences (voir le même tableau sous l'étiquette «`Cplex` appliqué à GAP »). Nous constatons que les résultats obtenus ne valent pas le temps qui leur est accordé. Bien que le temps imparti soit multiplié par 5, les résultats sont deux fois moins précis.

Bien que les résultats fournis par `Cplex` appliqués à RGAP soient satisfaisants, nous verrons que nous pouvons faire mieux en moins de temps de calcul en utilisant `ILS` vers RGAP.

TABLE 5.2 – Calcul des bornes inférieures et des bornes supérieures les plus mauvaises selon que la réduction des données est effectuée ou non.

| | | Cplex applied to GAP | | Cplex applied to RGAP | |
|---|---|---|---|---|---|
| | Optimal or best known | Time-limit $(m \times n)/10$ | Cost | Time-limit $(m\times)p/50$ | Cost |
| D10400 | 24961 † | 400 | 24976 | 53.80 | 24978 |
| D20400 | 24582 | 800 | 24621 | 112.80 | 24627 |
| D40400 | 24412 | 1600 | 24540 | 239.20 | 24456 |
| D15900 | 55412 | 1350 | 55429 | 198.00 | 55435 |
| D30900 | 54868 | 2700 | 54911 | 415.20 | 54898 |
| D60900 | 54605 | 5400 | 54768 | 915.60 | 54668 |
| D201600 | 97836 | 3200 | 97862 | 498.80 | 97867 |
| D401600 | 97113 | 6400 | 97232 | 1025.60 | 97182 |
| D801600 | 97052 | 12800 | 97406 | 2385.60 | 97166 |
| **Average** | | **3850.00** | **0.19%** | **649.40** | **0.10%** |

† Optimal value (Posta *et al.* [171])

### 5.4.4 Les résultats de l'ILS appliqués au RGAP

Dans une seconde phase, `ILS` est appliqué à l'instance RGAP et une fois qu'une solution est obtenue, elle est étendue à une solution GAP comme expliqué précédemment. Trois paramètres contrôlent la méta-heuristique `ILS` : le nombre d'itérations $N_{ILS}$, le temps limite $t_{ILS}$ accordé à `Cplex` pour trouver la solution initiale et le pourcentage $\vartheta$ des tâches impliquées dans la phase de perturbation. Ils ont été personnalisés grâce à un grand nombre d'essais expérimentaux. Les valeurs retenues pour ces paramètres sont : $t_{ILS} = m \times p/1000$ seconds, $N_{ILS} = 200{,}000$ itérations, et $\vartheta = 5\%$.

Les résultats de cette phase sont présentés dans le tableau 5.3. la quatrième et la troisième colonnes proposent respectivement le coût de la solution initiale et le temps qu'il a fallu pour l'obtenir. Les deux autres colonnes sont explicites.

Un dernier mot pour dire que l'ILS n'aurait pas été efficace s'il avait été appliqué directement aux instances GAP en raison de la faiblesse des deux voisinages explorés.

TABLE 5.3 – Résultats de l'ILS appliqué au RGAP

| | Optimal or best known | `Cplex` time-limit | Initial solution cost | Best solution cost | Total time |
|---|---|---|---|---|---|
| D10400 | 24961 | 2.69 | 24982 | 24975 | 22.31 |
| D20400 | 24582 | 5.64 | 24648 | 24633 | 22.91 |
| D40400 | 24412 | 11.96 | 24516 | 24455 | 70.02 |
| D15900 | 55412 | 9.90 | 55456 | 55434 | 65.28 |
| D30900 | 54868 | 20.76 | 55014 | 54897 | 162.37 |
| D60900 | 54605 | 45.78 | 54844 | 54662 | 332.86 |
| D201600 | 97836 | 24.94 | 97878 | 97867 | 232.34 |
| D401600 | 97113 | 51.28 | 97265 | 97167 | 475.60 |
| D801600 | 97052 | 119.28 | 97314 | 97107 | 989.63 |

### 5.4.5 Comparaison de l'ILS avec quatre méthodes concurrentes

Nous comparons `ILS` à trois méthodes récemment publiées et au solveur `Cplex` (voir le tableau 5.4 pour plus de détails). Il y a des articles plus récemment publiés (voir par exemple :Liu and Wang [128], Sadykov *et al.* [182], Sethanan and Pitakaso [191], Srivarapongse and Pijitbanjong [202]) mais malheureusement, ils ne testent pas l'ensemble de données bien connu des instances difficiles de type D et ne peuvent pas donc être utilisés pour la comparaison.

Étant donné que les temps de calcul des méthodes concurrentes n'étant pas directement comparables en raison des différents environnements utilisés, ils doivent être mis à l'échelle. Le SPEC (Standard Performance Evaluation Corporation) site Web `https://www.spec.org/cpu2006/results/cint2006.html` fournit une estimation de la vitesse d'exécution de la plupart des ordinateurs. Dans notre cas, une valeur Specint2006 est attachée à chaque machine dans l'avant-dernière colonne, plus la valeur est élevée, plus l'ordinateur est rapide. La dernière colonne donne les facteurs d'échelle. Par conséquent, si nous prenons un `Dell WorkStation` comme référence, nous voyons qu'un `Viglen CX130` (resp. `Dell Optiplex 390`) est à propos 1.33 (resp. 2.29) fois plus vite.

Les quatre méthodes concurrentes sont comparées à partir de la qualité de la solution dans le tableau 5.5, où la dernière ligne donne l'écart moyen par rapport à la meilleure. Il apparaît clairement que `HA18` est de loin le plus efficace, car il s'écarte du meilleur en moyenne d'environ 0.02% seulement, tandis que `ILS` est le deuxième meilleur (environ

TABLE 5.4 – Les méthodes concurrentes

| Label | Reference | Machine used | Specint values | Scaling factor |
|---|---|---|---|---|
| WW10 | [225] | Viglen CX130 (3.0 GHz) | 23.2 | 1.33 |
| MP08 | [138] | Dell Workstation (2.0 GHz) | 17.5 | 1 |
| HA18 | [93] | Dell Optiplex 390 (3.3GHz) | 40.1 | 2.29 |
| Cplex | | *Idem* | *Idem* | *Idem* |
| ILS | | *Idem* | *Idem* | *Idem* |

1, 5 fois plus précis que MP08, 2 fois plus précis que Cplex et environ 3fois plus précis que WW10).

TABLE 5.5 – Comparaison des méthodes concurrentes du point de vue de la qualité de la solution

| Instance | WW10 | MP08 | HA18 | Cplex | ILS |
|---|---|---|---|---|---|
| D10400 | 24976 | 24973 | 24964 | 24976 | 24975 |
| D20400 | 24631 | 24610 | 24582 | 24621 | 24633 |
| D40400 | 24572 | 24436 | 24412 | 24540 | 24455 |
| D15900 | 55462 | 55423 | 55412 | 55429 | 55434 |
| D30900 | 55012 | 54905 | 54888 | 54911 | 54897 |
| D60900 | 54785 | 54835 | 54605 | 54768 | 54662 |
| D201600 | 97921 | 97863 | 97836 | 97862 | 97867 |
| D401600 | 97328 | 97244 | 97142 | 97232 | 97167 |
| D801600 | 97449 | 97374 | 97095 | 97406 | 97107 |
| **Écart moyen par rapport au meilleur (%)** | 0.25 | 0.14 | **0.02**[1] | 0.19 | **0.09**[2] |

[1] HA18 est la méthode la plus précise.
[2] ILS est le plus est le deuxième meilleur. Il est environ 2 fois plus précis que Cplex, environ 1,5 fois mieux que le MP08 et environ 3 fois mieux que le WW10.

Le tableau 5.6 présente les temps de calcul des méthodes concurrentes. La colonne intitulée 'Time' rapporte les temps de calcul tirés des références et la colonne sous l'étiquette 'Scaled' propose les temps normalisés. À partir de la dernière ligne du tableau qui montre le temps normalisé moyen de chacune des méthodes concurrentes, nous pouvons observer que notre méthode est de loin la plus rapide, environ 14 (resp. 34,42 et 51) fois plus rapide que Cplex (resp. MP08, HA18 et WW10).

### 5.4.6 Discussion

La question la plus importante, après la phase expérimentale que nous avons menée, est probablement la suivante : pourquoi avons-nous obtenu de bons résultats malgré le fait

TABLE 5.6 – Comparaison des méthodes concurrentes du point de vue du temps de calcul

| | MP08 | WW10 | | HA18 | | Cplex | | ILS | |
|---|---|---|---|---|---|---|---|---|---|
| Instance | Time | Time | Scaled | Time | Scaled | Time | Scaled | Time | Scaled |
| D10400 | 3000 | 10000 | 13300 | 1082.33 | 2478.54 | 400 | 916.00 | 22.31 | 51.09 |
| D20400 | | | | 2260.80 | 5177.23 | 800 | 1832.00 | 37.98 | 86.97 |
| D40400 | | | | 4791.56 | 10972.67 | 1600 | 3664.00 | 70.02 | 160.37 |
| D15900 | 10000 | 10000 | 13300 | 3297.39 | 7551.02 | 1350 | 3091.50 | 65.28 | 149.49 |
| D30900 | | | | 7129.18 | 16325.82 | 2700 | 6183.00 | 162.37 | 371.83 |
| D60900 | | | | 17100.89 | 39161.04 | 5400 | 12366.00 | 332.86 | 762.25 |
| D201200 | 50000 | 50000 | 66500 | 5116.43 | 11716.62 | 3200 | 7328.00 | 232.34 | 532.06 |
| D401200 | | | | 12701.02 | 29085.34 | 6400 | 14656.00 | 475.60 | 1089.12 |
| D801200 | | | | 46477.26 | 106432.93 | 12800 | 29312.00 | 989.63 | 2266.25 |
| **Average** | **21000 s** | | **31033 s** | | **25433 s** | | **8817 s** | | **608** $s$[1] |

[1] L'ILS est la méthode la plus rapide. Elle est environ 14 fois plus rapide que Cplex, environ 34 fois plus rapide que MP08, environ 42 fois plus rapide que HA18 et environ 51 fois plus rapide que WW10.

que les deux quartiers à petite échelle proposés sont faibles ? La réponse est la suivante : les bons résultats obtenus sont dus à la phase de réduction des données.

Un argument formel est le suivant. Pour une instance GAP, sans phase de réduction des données, nous définissons deux valeurs, la pire borne inférieure $l_{GAP} = \sum_{j \in N} min_{i \in M} c_{ij}$ et la pire borne supérieure $u_{GAP} = \sum_{j \in N} max_{i \in M} c_{ij}$. Après avoir appliqué la procédure de réduction des données, laisser faire $l_{RGAP} = \sum_{j \in P} min_{i \in I_j} c_{ij}$ et laisser faire $u_{RGAP} = \sum_{j \in P} max_{i \in I_j} c_{ij}$. Pour la même instance GAP, si la réduction des données est appliquée, la pire borne inférieure devient $pcost + l_{RGAP}$ et la pire borne supérieure devient $pcost + u_{RGAP}$.

Ces pures bornes sont présentées dans le tableau 5.7. Nous observons que les bornes sont plus serrées après avoir effectuée la réduction des données. Par exemple, nous constatons que, sans réduction de données, le coût de toute affectation réalisable pour l'instance D10400 se situe dans l'intervalle [7218,41222], mais après avoir appliqué la réduction de données, l'intervalle sera resserré à [18400,29532]. Par conséquent, l'heuristique de réduction des données ne conserve que les sélections de tâches /machines «prometteuses» et réduit les autres. C'est le résultat le plus important de la procédure de réduction des données et cela explique pourquoi il est préférable de traiter les instances RGAP plutôt que les instances GAP. Ce fait explique aussi pourquoi toute méthode donnera des résultats plus rapides et plus précis lorsqu'elle est appliqué à des instances RGAP si elle était appliquée sans réduction de données à des instances GAP.

TABLE 5.7 – Comparaison des mauvaises bornes inférieures et supérieures selon que la réduction des données est effectuée ou non.

| | Worst bounds without data reduction | | Worst bounds after data reduction | |
|---|---|---|---|---|
| | $l_{GAP}$ | $u_{GAP}$ | $l_{RGAP} + pcost$ | $u_{RGAP} + pcost$ |
| D10400 | 7218 | 41222 | 18400 | 29532 |
| D20400 | 5244 | 43332 | 18031 | 29395 |
| D40400 | 3723 | 44926 | 16479 | 30149 |
| D15900 | 12974 | 95869 | 38060 | 65657 |
| D30900 | 9383 | 99560 | 37302 | 66331 |
| D60900 | 6496 | 102225 | 31893 | 67400 |
| D201600 | 20689 | 173695 | 62802 | 117154 |
| D401600 | 14454 | 179034 | 60663 | 117690 |
| D801600 | 10390 | 183367 | 47634 | 126380 |

## 5.5 Conclusion

Une méthode en deux phases est proposée pour GAP. En relaxant les contraintes du sac à dos, l'étape de réduction des données utilise l'optimisation par sous-gradient pour réduire la taille de l'instance GAP. En pratique, cela permet de supprimer jusqu'à 96% des variables de GAP. Le GAP réduit obtenu par cette procédure est petit et peu dense. De plus, sa solution optimale peut être facilement étendue à une solution optimale ou quasi-optimale de GAP.

La deuxième étape exécute `ILS` sur l'instance GAP réduite et l'ensemble de la méthode est testé sur un benchmark largement utilisé d'instances larges et difficiles. Le résultat est que la méta-heuristique fournit des solutions GAP de bonne qualité en un temps très court. C'est exactement ce que l'on attend d'elle, bien que son défaut soit de dépendre d'un solveur commercial pour trouver des solutions initiales à la procédure de réduction des données ainsi qu'à la méta-heuristique ILS.

Bien qu'elle soit autosuffisante et constitue une alternative satisfaisante aux méthodes concurrentes pour GAP, l'étude de cette section est une recherche préliminaire pour aborder un problème plus complexe : le problème d'allocation de mémoire dans un système embarqué évoqué dans la section 5.1.2.

# Conclusion générale

On a étudié le problème d'allocation mémoire dans les systèmes embarqué (MAP), c'est un problème d'optimisation combinatoire NP-difficile.

Une introduction à l'optimisation combinatoire et à la théorie de la complexité est présenté dans le chapitre 1 qui sert à situer le contexte de cette thèse. On a ensuite présenté la panoplie des techniques utilisées dans notre étude au chapitre 2. Suivie d'une description sur des techniques des optimisations de la mémoire pour les systèmes embarqués dans le chapitre 3. Dans le chapitre 4 nous avons expliqué le problème de l'allocation de mémoire dans les systèmes embarqué, où un état de l'art est présent. Dans les trois chapitres suivants, on a détaillé nos contributions.

Le chapitre 5 présente le problème d'affectation généralisé (GAP), certaines de ses applications pratiques et propriétés, ont motivé notre approche proposée (comment le modèle MAP devient un GAP). La méthode proposée dans cette recherche commence par une heuristique de réduction de données pour réduire la taille de GAP pour construire un très petit problème qui sera résolu facilement et très rapidement par un solveur MIP (`Cplex`). La méthode proposée est testée sur l'ensemble des instances. Puis appliquerons la recherche locale ILS combinant réduction des données pour obtenir un sortant qui sera amélioré par la recherche locale en utilisant la matrice de coût d'origine.

Voici quelques domaines qui pourraient faire l'objet de recherches futures :

— Montrer que nos approches sont génériques, et peuvent être appliquées au problème général de l'optimisation combinatoire.

— Il serait intéressant de tester nos approches avec d'autres bases de données et sous d'autres contraintes.

— Les problèmes multi-objectives ne sont pas traités dans ce travail et pourraient être incorporés pour les futurs travaux.

# Publications de l'auteur

## Publications internationales

1. Fatima Guessoum, Salim Haddadi, **Gattal Elhachemi** (2019), Simple, Yet Fast and Effective Two-Phase Method for Nurse Rostering, American Journal of Mathematical and Management Sciences, published online : https://dx.doi.org/10.1080/01966324.2019.1570882.
2. **Gattal Elhachemi**, Salim Haddadi (2021), Combining data reduction, MIP solver and iterated local search for generalized assignment, International Journal of Management Science and Engineering Management TMSE, published online : https://doi.org/10.1080/17509653.2021.1970039.

# Bibliographie


[1] Behrouz Afshar-Nadjafi. A solution procedure for preemptive multi-mode project scheduling problem with mode changeability to resumption. *Applied computing and informatics*, 14(2) :192–201, 2018.

[2] R Agarwal. *Solving parallel machine scheduling problems with variable depth local search*. PhD thesis, School of Mathematics, University of Southampton, 2004.

[3] R K Ahuja, J B Orlin, and D Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91(1) :71–97, 2001.

[4] R K Ahuja, Ö Ergun, J B Orlin, and A P Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1) :75–102, 2002.

[5] R K Ahuja, J B Orlin, and D Sharma. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters*, 31(3) :185–194, 2003.

[6] R K Ahuja, Ö Ergun, J B Orlin, and A P Punnen. Very large-scale neighborhood search : Theory, algorithms, and applications. *Handbook of Approximation Algorithms and Metaheuristics*, 10, 2007.

[7] R K Ahuja, K C Jha, J B Orlin, and D Sharma. Very large-scale neighborhood search for the quadratic assignment problem. *Informs journal on computing*, 19(4) :646–657, 2007.

[8] S Akyol and B Alatas. Plant intelligence based metaheuristic optimization algorithms. *Artificial Intelligence Review*, 47(4) :417–462, 2017.

[9] F Angiolini, L Benini, and A Caprara. An efficient profile-based algorithm for scratch-pad memory partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(11) :1660–1676, 2005.

[10] Claudia Archetti, Maria Grazia Speranza, and Alain Hertz. A tabu search algorithm for the split delivery vehicle routing problem. *Transportation science*, 40(1) :64–73, 2006.

[11] Antonio Artes, Jose L Ayala, Jos Huisken, and Francky Catthoor. Survey of low-energy techniques for instruction memory organisations in embedded systems. *Journal of Signal Processing Systems*, 70(1) :1–19, 2013.

[12] G. Ascia, V. Catania, A.G. Di Nuovo, M. Palesi, and D. Patti. Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems. *Applied Soft Computing*, 11(1) :382–398, 2011.

[13] P Avella, M Boccia, and I Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45(3) :543–555, 2008.

[14] O. Avissar, R. Barua, and D. Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Transactions on Embedded Computing Systems*, 1(1) :6–26, 2002.

[15] F. Balasa, H. Zhu, and I.I. Luican. Computation of storage requirements for multi-dimensional signal processing applications. *IEEE Transactions on Very Large Scale Integration Systems*, 15(4) :447–460, 2007.

[16] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory : A design alternative for cache on-chip memory in embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 73–78, New York, NY, USA, 2002. ACM.

[17] J Barbas and Á Marín. Maximal covering code multiplexing access telecommunication networks. *European Journal of Operational Research*, 159(1) :219–238, 2004.

[18] J Beasley. Lagrangian relaxation. In Colin R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, chapter 6, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[19] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8) :716, 1952.

[20] L. Benini, A. Macii, and M. Poncino. A recursive algorithm for low-power memory partitioning. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, ISLPED '00, pages 78–83, New York, NY, USA, 2000. ACM.

[21] L. Benini, D. Bruni, A. Macii, and E. Macii. Hardware-assisted data compression for energy minimization in systems with embedded processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '02, pages 449–454, Washington, DC, USA, 2002. IEEE Computer Society.

[22] L. Benini, L. Macchiarulo, A. Macii, and M. Poncino. Layout-driven memory synthesis for embedded systems-on-chip. *IEEE Transactions on Very Large Scale Integration Systems*, 10(2) :96–105, 2002.

[23] Luca Benini, Alberto Macii, and Massimo Poncino. Energy-aware design of embedded memories : A survey of technologies, architectures, and optimization techniques. *ACM Transactions on Embedded Computing Systems*, 2(1) :5–32, 2003.

[24] Ala-Eddine Benrazek, Zineddine Kouahla, Brahim Farou, Mohamed Amine Ferrag, Hamid Seridi, and Muhammet Kurulay. An efficient indexing for internet of things massive data based on cloud-fog computing. *Transactions on emerging telecommunications technologies*, 31(3) :e3868, 2020.

[25] Jacek Blazewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Scheduling computer and manufacturing processes*. springer science & Business media, 2013.

[26] O Borzog-Haddad, M Solgi, and H A Loiciga. *Metaheuristic and evolutionary algorithms for engineering optimization*, volume 294. John Wiley & Sons, 2017.

[27] Hector Cancela Bosi. *Evaluation de la surete de fonctionnement : modeles combinatoires et markoviens*. PhD thesis, 1996.

[28] I Boussaid. *de métaheuristiques pour l'optimisation continue*. PhD thesis, Université Paris-Est ; Université des sciences et de la technologie Houari Boumédienne (Alger), 2013.

[29] A Brabazon, M O'Neill, and S McGarraghy. Plant-inspired algorithms. In *Natural Computing Algorithms*, pages 455–477. Springer, 2015.

[30] RG Busacker and PJ Gowen. A procedure for determining a family of minimum-cost network flow patterns, oro technical paper 15. *Operational Research Office, Johns Hopkins University, Baltimore*, 11, 1961.

[31] G M Campbell and M Diaby. Development and evaluation of an assignment heuristic for allocating cross-trained workers. *European Journal of Operational Research*, 138(1) :9–20, 2002.

[32] J F Campbell and A Langevin. The snow disposal assignment problem. *Journal of the Operational Research Society*, 46(8) :919–929, 1995.

[33] J Carlier and Ph Chretienne. Un domaine très ouvert : les problèmes d'ordonnancement. *RAIRO-Operations Research-Recherche Opérationnelle*, 16(3) :175–217, 1982.

[34] R. C. Carlson and G. L. Nemhauser. Scheduling to Minimize Interaction Cost. *Operations Research*, 14(1) :52–58, February 1966.

[35] Arthur E Carter and Cliff T Ragsdale. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European journal of operational research*, 175(1) :246–257, 2006.

[36] D Cattrysse, Z Degraeve, and J Tistaert. Solving the generalised assignment problem using polyhedral results. *European Journal of Operational Research*, 108(3) :618–628, 1998.

[37] Anantha P Chandrakasan, Miodrag Potkonjak, Renu Mehra, Jan Rabaey, and Robert W Brodersen. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1) :12–31, 1995.

[38] C Chekuri and S Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3) :713–728, 2005.

[39] M Cheraitia, S Haddadi, and A Salhi. Hybridising plant propagation and local search for uncapacitated exam scheduling problems. *International Journal of Services and Operations Management*, 32(4) :450–467, 2019.

[40] Nawal Cherfi and Mhand Hifi. A column generation method for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 46(1) :51–73, 2010.

[41] D. Cho, S. Pasricha, I. Issenin, N. D. Dutt, M. Ahn, and Y. Paek. Adaptive scratch pad memory management for dynamic behavior of multimedia applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(4) :554–567, 2009.

[42] N Christofides, A Mingozzi, P Toth, and C Sandi. Chapter 11. *Combinatorial Optimization. John Willey, Chichester*, 1979.

[43] H Christos. Papadimitriou : Computational complexity. *Addison-Wesley*, 2(3) :4, 1994.

[44] P C Chu and J E Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1) :17–23, 1997.

[45] Fan RK Chung, Michael R Garey, and David S Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods*, 3(1) :66–76, 1982.

[46] Maurice Clerc and Patrick Siarry. Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires. *J3eA*, 3 :007, 2004.

[47] B Codenotti, G Manzini, L margara, and G Resta. Perturbation : an efficient technique for the solution of very large instances of the euclidean tsp. *INFORMS J Comput*, 8(2) :125–133, 1996.

[48] J. Cong, W. Jiang, B. Liu, and Y. Zou. Automatic memory partitioning and scheduling for throughput and power optimization. *ACM Transactions on Design Automation of Electronic Systems*, 16(2) :15 :1–15 :25, 2011.

[49] R G Cromley and D M Hanink. Coupling land use allocation models with raster GIS. *Journal of Geographical Systems*, 1(2) :137–153, 1999.

[50] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4) :393–410, 1954.

[51] George B Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[52] Richard Dawkins. The selfish gene new york : Oxford university press. *DawkinsThe Selfish Gene1976*, 1976.

[53] D.Du and P.M Pardalos. *Handbbok of combinatorial optimization.* Springer, 2007.

[54] F Debbat and FT Bendimerad. Les métaheuristiques : application réseaux intelligents d'antennes. *Tendances des Applications Mathématiques*, page 389.

[55] V G Deineko and G J Woeginger. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical programming*, 87(3) :519–542, 2000.

[56] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2) :342–354, 1992.

[57] J.F. Deverge and I. Puaut. WCET-directed dynamic scratchpad memory allocation of data. In *19th Euromicro Conference on Real-Time Systems*, pages 179–190, 2007.

[58] Ronald A DeVore and Vladimir N Temlyakov. Some remarks on greedy algorithms. *Advances in computational Mathematics*, 5(1) :173–187, 1996.

[59] J A Díaz and E Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1) :22–38, 2001.

[60] G Dobson and R S Nambimadom. The batch loading and scheduling problem. *Operations Research*, 49(1) :52–65, 2001.

[61] Michael Dom. *Recognition, Generation, and Application of Binary Matrices with the Consecutive Ones Property.* Cuvillier Gottingen, 2009.

[62] Marco Dorigo and Christian Blum. Ant colony optimization theory : A survey. *Theoretical computer science*, 344(2-3) :243–278, 2005.

[63] U Dorndorf and E Pesch. Fast clustering algorithms. *ORSA Journal on Computing*, 6(2) :141–153, 1994.

[64] Andreas Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40(1) :1–8, 1988.

[65] Stuart Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1) :48–51, 2002.

[66] F Dusberger and G R Raidl. A variable neighborhood search using very large neighborhood structures for the 3-staged 2-dimensional cutting stock problem. In *International Workshop on Hybrid Metaheuristics*, pages 85–99. Springer, 2014.

[67] B. Egger, J. Lee, and H. Shin. Dynamic scratchpad memory management for code in portable systems with an MMU. *ACM Transactions on Embedded Computing Systems*, 7(2) :1–38, 2008.

[68] L. Epstein and R. van Stee. Improved results for a memory allocation problem. *Theory of Computing Systems*, 48(1) :79–92, 2011.

[69] Ö Ergun and J B Orlin. A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem. *Discrete Optimization*, 3(1) :78–85, 2006.

[70] M L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27 :1–18, 1981.

[71] M L Fisher. An applications oriented guide to lagrangian relaxation. *Interfaces*, 15 :10–21, 1985.

[72] L R Foulds and J M Wilson. A variation of the generalized assignment problem arising in the New Zealand dairy industry. *Annals of Operations Research*, 69 :105–114, 1997.

[73] Giorgio Gallo and Bruno Simeone. On the supermodular knapsack problem. *Mathematical Programming*, 45(1) :295–309, 1989.

[74] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[75] Michael R Garey. Computers and intractability : A guide to the theory of np-completeness. *Revista Da Escola De Enfermagem Da USP*, 44(2) :340, 1979.

[76] R Geleijn, M van der Meer, Q Van der Post, and D van der Berg. The plant propagation algorithm on timetables : First results. *EVO* 2019*, page 2, 2019.

[77] F Gembicki and Y Haimes. Approach to performance and sensitivity multiobjective optimization : The goal attainment method. *IEEE Transactions on Automatic control*, 20(6) :769–771, 1975.

[78] A M Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, 2 :82–114, 1974.

[79] A Ghoniem, T Flamand, and M Haouari. Exact solution methods for generalized assignment problem with location/allocation considerations. *INFORMS Journal on Computing*, 28(3) :589–602, 2016.

[80] A Ghoniem, T Flamand, and M Haouari. Optimisation-based very large-scale neighborhood search for generalized assignment problems with location/allocation considerations. *INFORMS Journal on Computing*, 28(3) :575–588, 2016.

[81] Saurav Kumar Ghosh, P Vishnuvardhan, Satya Gautam Vadlamudi, Aritra Hazra, Soumyajit Dey, and Partha Pratim Chakrabarti. Relspec : a framework for reliability aware design of component based embedded systems. *Design Automation for Embedded Systems*, 21(1) :37–87, 2017.

[82] F Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549, 1986.

[83] Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008*, pages 77–103. Springer, 2010.

[84] Michel Gondran and Michel Minoux. *Graphes et algorithmes*, volume 79. Eyrolles Paris, 1979.

[85] F Guessoum, S Haddadi, and E Gattal. Simple, yet fast and effective two-phase method for nurse rostering. *American Journal of Mathematical and Management Sciences*, 39(1) :1–19, 2020.

[86] G Gutin. Ap punnen the travelling salesman problem and its variants g. *Kluwer Academic Publishers, Secaucus, NJ, USA*, 2002.

[87] S Haddadi and H Ouzia. Effective algorithm and heuristic for the generalized assignment problem. *European Journal of Operational Research*, 153(1) :184–190, 2004.

[88] Salim Haddadi and Hacene Ouzia. Effective algorithm and heuristic for the generalized assignment problem. *European Journal of Operational Research*, 153(1) :184–190, 2004.

[89] S Haddadi, M Cheraitia, and A Salhi. A two-phase heuristic for set covering. *International Journal of Mathematics in Operational Research*, 13(1) :61–78, 2018.

[90] S Haddadi. Lagrangian decomposition based heuristic for the generalized assignment problem. *INFOR : Information Systems and Operational Research*, 37(4) :392–402, 1999.

[91] Salim Haddadi. Lagrangian decomposition based heuristic for the generalized assignment problem. *INFOR : Information Systems and Operational Research*, 37(4) :392–402, 1999.

[92] S Haddadi. Three-phase method for nurse rostering. *International Journal of Management Science and Engineering Management*, 14(3) :193–205, 2019.

[93] S Haddadi. Variable-fixing then subgradient optimization guided very large scale neighborhood search for the generalized assignment problem. *4OR*, 17(3) :261–295, 2019.

[94] S Haddadi. Hybrid metaheuristic for generalised assignment. *International Journal of Innovative Computing and Applications*, 11(1) :46–60, 2020.

[95] Michael Hahsler and Kurt Hornik. Tsp-infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2) :1–21, 2007.

[96] Abdelaziz I Hammouri, Malik Sh Braik, Mohammed Azmi Al-Betar, and Mohammed A Awadallah. Isa : a hybridization between iterated local search and simulated annealing for multiple-runway aircraft landing problem. *Neural Computing and Applications*, pages 1–21, 2019.

[97] M Held and R M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[98] Mhand Hifi, Mustapha Michrafy, and Abdelkader Sbihi. Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *Journal of the Operational Research Society*, 55(12) :1323–1332, 2004.

[99] Mhand Hifi, Mustapha Michrafy, and Abdelkader Sbihi. A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 33(2-3) :271–285, 2006.

[100] A J Higgins. Optimizing cane supply decisions within a sugar mill region. *Journal of Scheduling*, 2(5) :229–244, 1999.

[101] S C Ho and W Szeto. A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. *Transportation Research Part B : Methodological*, 95 :340–363, 2017.

[102] JH Holland. Adaptation in natural and artificial systems, univ. of mich. press. *Ann Arbor*, 1975.

[103] L. Idoumghar, M. Idrissi Aouad, M. Melkemi, and R. Schott. Metropolis particle swarm optimization algorithm with mutation operator for global optimization problems. In *22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 35–42, 2010.

[104] L. Idoumghar, M. Melkemi, R. Schott, and M. Idrissi Aouad. Hybrid PSO-SA type algorithms for multimodal function optimization and reducing energy consumption in embedded systems. *Applied Computational Intelligence and Soft Computing*, 2011 :3 :1–3 :12, 2011.

[105] M. Idrissi Aouad, R. Schott, and O. Zendra. A tabu search heuristic for scratch-pad memory management. In *International Conference on Software Engineering and Technology*, volume 64, pages 386–390, Rome, Italy, 2010. World Academy of Science, Engineering and Technology.

[106] M. Idrissi Aouad, R. Schott, and O. Zendra. Hybrid heuristics for optimizing energy consumption in embedded systems. In E. Gelenbe, R. Lent, G. Sakellari, A. Sacan, H. Toroslu, and A. Yazici, editors, *Computer and Information Sciences*, pages 409–414, Dordrecht, 2010. Springer Netherlands.

[107] I. Issenin, E. Brockmeyer, M. Miranda, and N. Dutt. DRDU : A data reuse analysis technique for efficient scratch-pad memory management. *ACM Transactions on Design Automation of Electronic Systems*, 12(2), 2007.

[108] Marija Ivanovic, Aleksandar Savić, Dragan Urošević, and Djordje Dugošija. A new variable neighborhood search approach for solving dynamic memory allocation problem. *Yugoslav Journal of Operations Research*, 2018. Published online : https://doi.org/10.2298/YJOR161015018I.

[109] K C Jha. *Very large-scale neighborhood search heuristics for combinatorial optimization problems*. PhD thesis, University of Florida, 2004.

[110] L Jourdan. *Métaheuristiques pour l'extraction de connaissances : Application à la génomique*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2003.

[111] M. Kandemir, M.J. Irwin, G. Chen, and I. Kolcu. Compiler-guided leakage optimization for banked scratch-pad memories. *IEEE Transactions on Very Large Scale Integration Systems*, 13(10) :1136–1146, 2005.

[112] S.D. Khan and H. Shin. Effective memory access optimization by memory delay modeling, memory allocation, and buffer allocation. In *2009 International SoC Design Conference (ISOCC)*, pages 153–156, 2009.

[113] N. Kim and R. Peng. A memory allocation and assignment method using multiway partitioning. In *IEEE International SOC Conference, 2004. Proceedings.*, pages 143–144, 2004.

[114] Yongjoo Kim, Jongeun Lee, Aviral Shrivastava, and Yunheung Paek. Operation and data mapping for cgras with multi-bank memory. *ACM Sigplan Notices*, 45(4) :17–26, 2010.

[115] S Kirkpatrick, C D Gelatt, M P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[116] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[117] Hakduran Koc, Ozcan Ozturk, M Kandemir, Sri Hari Krishna Narayanan, and Ehat Ercanli. Minimizing energy consumption of banked memories using data recomputation. In *Proceedings of the 2006 international symposium on Low power electronics and design*, pages 358–362. ACM, 2006.

[118] T.S.R. Kumar, C.P. Ravikumar, and R. Govindarajan. MAX : A multi objective memory architecture exploration framework for embedded systems-on-chip. In *20th*

*International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, pages 527–533, 2007.

[119] J Kytöjoki, T Nuortio, O Bräysy, and M Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & operations research*, 34(9) :2743–2757, 2007.

[120] François Laburthe. *Contraintes et algorithmes en optimisation combinatoire*. PhD thesis, Paris 7, 1998.

[121] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28 :497, 1960.

[122] Gilbert Laporte and Yves Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5(2) :77–85, 1983.

[123] Eugene L Lawler and David E Wood. Branch-and-bound methods : A survey. *Operations research*, 14(4) :699–719, 1966.

[124] W. H. Lee and M. Chang. A study of dynamic memory management in C++ programs. *Computer Languages, Systems & Structures*, 28(3) :237–272, 2002.

[125] Adam N Letchford, Jens Lysgaard, and Richard W Eglese. A branch-and-cut algorithm for the capacitated open vehicle routing problem. *Journal of the Operational Research Society*, 58(12) :1642–1651, 2007.

[126] Y Li, Y Tao, and F Wang. A compromised large-scale neighborhood search heuristic for capacitated air cargo loading planning. *European Journal of Operational Research*, 199(2) :553–560, 2009.

[127] Y Li, H Chen, and C Prins. Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, 252(1) :27–38, 2016.

[128] Yan Y Liu and Shaowen Wang. A scalable parallel genetic algorithm for the generalized assignment problem. *Parallel computing*, 46 :98–119, 2015.

[129] X Liu, G Laporte, Y Chen, and R He. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers & Operations Research*, 2017.

[130] P Lopez and F Roubellat. éditeurs. *Ordonnancement de la production. Hermes Science Publications, Paris*, 2001.

[131] H R Lourenço, O C Martin, and T Stützle. Iterated local search : Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer, 2019.

[132] M l Marmion. *Recherche locale et optimisation combinatoire : de l'analyse structurelle d'un problème à la conception d'algorithmes efficaces*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2011.

[133] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3) :414–424, 1999.

[134] Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2) :325–332, 2000.

[135] S. Meftali, F. Gharsalli, F. Rousseau, and A.A. Jerraya. An optimal memory allocation for application-specific multiprocessor system-on-chip. In *Proceedings of the 14th International Symposium on Systems Synthesis*, ISSS '01, pages 19–24, New York, NY, USA, 2001. ACM.

[136] C Meyers and J B Orlin. Very large-scale neighborhood search techniques in timetabling problems. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 24–39. Springer, 2006.

[137] George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3) :284–304, 1980.

[138] Snežana Mitrović-Minić and Abrahim P. Punnen. Very large-scale variable neighborhood search for the generalized assignment problem. *Journal of Interdisciplinary Mathematics*, 11(5) :653–670, 2008.

[139] B. Mohammad, P. Bassett, J. Abraham, and A. Aziz. Cache organization for embeded processors : CAM-vs-SRAM. In *2006 IEEE International SOC Conference*, pages 299–302, 2006.

[140] M A S Monfared and M Etemadi. The impact of energy function structure on solving generalized assignment problem using hopfield neural network. *European Journal of Operational Research*, 168(2) :645–654, 2006.

[141] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826 :1989, 1989.

[142] Martin Moser, Dusan P Jokanovic, and Norio Shiratori. An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(3) :582–589, 1997.

[143] Mohamed Elhavedh Ould Ahmed Mounir. *Contribution à la résolution du sac-à-dos à contraintes disjonctives*. PhD thesis, 2009.

[144] T.R. Mück and A.A. Fröhlich. Run-time scratch-pad memory management for embedded systems. In *37th Annual Conference of the IEEE Industrial Electronics Society*, IECON 2011, pages 2833–2838, 2011.

[145] L. Nachtergaele, F. Catthoor, and C. Kulkarni. Random-access data storage components in customized architectures. *IEEE Design & Test of Computers*, 18(3) :40–54, 2001.

[146] S Nag. Adaptive plant propagation algorithm for solving economic load dispatch problem. *arXiv preprint arXiv :1708.07040*, 2017.

[147] R M Nauss. Solving the generalized assignment problem : an optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3) :249–266, 2003.

[148] George L Nemhauser and Laurence A Wolsey. Integer and combinatorial optimization john wiley & sons. *New York*, 118, 1988.

[149] G L Nemhauser. The age of optimization : Solving large-scale real-world problems. *Operations Research*, 42 :5–14, 1994.

[150] Sandra Ulrich Ngueveu. *Problèmes de tournées de véhicules avec contraintes particulières pour la maîtrise des risques*. PhD thesis, PhD thesis, Université de Technologie de Troyes, 2009.

[151] T. Noergaard. *Embedded systems architecture – A comprehensive guide for engineers and programmers*. Newnes, Burlington, MA, USA, 2005.

[152] J Nowakovski, W Schwärzler, and E Triesch. Using the generalized assignment problem in scheduling the ROSAT space telescope. *European Journal of Operational Research*, 112(3) :531–541, 1999.

[153] Cristian Oliva, Philippe Michelon, and Christian Artigues. Constraint and linear programming : Using reduced costs for solving the zero/one multiple knapsack problem. In *Proc. of the Workshop on Cooperative Solvers in Constraint Programming (CoSolv 01), Paphos, Cyprus*, pages 87–98, 2001.

[154] T Öncan. A survey of the generalized assignment problem and its applications. *INFOR : Information Systems and Operational Research*, 45(3) :123–141, 2007.

[155] L Özbakir, A Baykasoğlu, and P Tapkan. Bees algorithm for generalized assignment problem. *Applied Mathematics and Computation*, 215(11) :3782–3795, 2010.

[156] O. Ozturk, G. Chen, M. Kandemir, and I. Kolcu. Compiler-guided data compression for reducing memory consumption of embedded applications. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 814–819, Piscataway, NJ, USA, 2006. IEEE Press.

[157] O Ozturk, M Kandemir, and M J Irwin. Using data compression for increasing memory system utilization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(6) :901–914, 2009.

[158] M Paauw and D Van Den Berg. Paintings, polygons and plant propagation. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 84–97. Springer, 2019.

[159] P.R. Panda, N.D. Dutt, and A. Nicolau. On-Chip vs. Off-Chip memory : The data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automation of Electronic Systems*, 5(3) :682–704, 2000.

[160] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 6(2) :149–206, 2001.

[161] P.R. Panda, N.D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C. Kulkarni, and E. De Greef. Data memory organization and optimizations in application-specific systems. *IEEE Design & Test of Computers*, 18(3) :56–68, 2001.

[162] C H Papadimitriou and K Steiglitz. *Combinatorial optimization : algorithms and complexity*. Courier Corporation, 1982.

[163] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization : algorithms and complexity*. Courier Corporation, 1998.

[164] C H Papadimitriou. *The complexity of combinatorial optimization problems*. PhD thesis, Princeton University, 1976.

[165] C H Papadimitriou. *Computational Complexity*. Prentice-Hall, 1994.

[166] CH Papadimitriou. Computational complexity addison-wesley reading. *Massachusetts Google Scholar*, 1994.

[167] Panos M Pardalos, Ding-Zhu Du, and Ronald L Graham. *Handbook of combinatorial optimization*. Springer, 2013.

[168] A Pigatti, M P de Aragão, and E Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. *Electronic Notes in Discrete Mathematics*, 19 :389–395, 2005.

[169] F. Poletti, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J.M. Mendias. An integrated hardware/software approach for run-time scratchpad management. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 238–243, New York, NY, USA, 2004. ACM.

[170] D Porumbel. Dimacs graphs : Benchmark instances and best upper bounds. http://cedric.cnam.fr/~porumbed/graphs/, 2009. Accessed : 2021-03-15.

[171] M Posta, J A Ferland, and P Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52(3) :629–644, 2011.

[172] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & operations research*, 31(12) :1985–2002, 2004.

[173] P Putz. Subgradient optimization based lagrangian relaxation and relaxand-cut approaches for the bounded diameter minimum spanning tree problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2007.

[174] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits : a design perspective*, volume 7. Pearson education Upper Saddle River, NJ, 2003.

[175] A. Ramachandran and M.F. Jacome. Xtream-fit : an energy-delay efficient data memory subsystem for embedded media processing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6) :832–848, 2005.

[176] Marc Reimann, Karl Doerner, and Richard F Hartl. D-ants : Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4) :563–591, 2004.

[177] Jacques Renaud, Gilbert Laporte, and Fayez F Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23(3) :229–235, 1996.

[178] G M Ribeiro and G Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & operations research*, 39(3) :728–735, 2012.

[179] S Ropke and D Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4) :455–472, 2006.

[180] Bernard Roy and Daniel Carré. *Les problèmes d'ordonnancement : applications et méthodes*, volume 2. Dunod, 1964.

[181] K S Ruland. A model for aeromedical routing and scheduling. *International Transactions in Operational Research*, 6(1) :57–73, 1999.

[182] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, and Eduardo Uchoa. Column generation based heuristic for the generalized assignment problem. *XLVII Simpósio Brasileiro de Pesquisa Operacional, Porto de Galinhas, Brazil*, pages 3624–3631, 2015.

[183] A Salhi and E S Fraga. Nature-inspired optimisation approaches and the new plant propagation algorithm. In *Proc. of the International Conference on Numerical Analysis and Optimization*, pages K2.1–K2.8. 2011.

[184] Michel Salomon. *Etude de la parallésation de méthodes heuristiques d'optimisation combinatoire*. PhD thesis, Thèse de doctorat, Université Luis Pasteur Strasbourg 1, 2001.

[185] J. Sánchez-Oro, M. Sevaux, A. Rossi, R. Martí, and A Duarte. Improving the performance of embedded systems with variable neighborhood search. *Applied Soft Computing*, 53 :217–226, 2017.

[186] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1) :17–29, 1995.

[187] M Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6) :831–841, 1997.

[188] Abdelkader Sbihi. *Les méthodes hybrides en optimisation combinatoire : algorithmes exacts et heuristiques*. PhD thesis, 2003.

[189] B I Selamoglu and A Salhi. The plant propagation algorithm for discrete optimisation : The case of the travelling salesman problem. In *Nature-Inspired Computation in Engineering*, pages 43–61. Springer Nature, 2016.

[190] M. Serrano and H.-J. Boehm. Understanding memory allocation of scheme programs. *ACM SIGPLAN Notices*, 35(9) :245–256, 2000.

[191] Kanchana Sethanan and Rapeepan Pitakaso. Improved differential evolution algorithms for solving generalized assignment problem. *Expert Systems with Applications*, 45 :450–459, 2016.

[192] M. Sevaux, A. Rossi, M. Soto, A. Duarte, and R. Marti. GRASP with ejection chains for the dynamic memory allocation in embedded systems. *Soft Computing*, 18 :1515–1527, 2014.

[193] Zili Shao, Yongpan Liu, Yiran Chen, and Tao Li. Utilizing PCM for energy optimization in embedded systems. In *IEEE Computer Society Annual Symposium on VLSI*, ISVLSI'12, pages 398–403. IEEE, 2012.

[194] J F Shapiro. A survey of lagrangean techniques for discrete optimization. In *Annals of Discrete Mathematics*, volume 5, pages 113–138. Elsevier, 1979.

[195] W T Shiue and C Chakrabarti. Memory design and exploration for low power, embedded systems. *Journal of VLSI Signal Processing*, 29(3) :167–178, 2001.

[196] K. Shyam and R. Govindarajan. An array allocation scheme for energy reduction in partitioned memory architectures. In S. Krishnamurthi and M. Odersky, editors, *Compiler Construction*, pages 32–47, Berlin, Germany, 2007. Springer.

[197] M. Singh and V.K. Prasanna. Algorithmic techniques for memory energy reduction. In K. Jansen, M. Margraf, M. Mastrolilli, and J.D.P. Rolim, editors, *Experimental and Efficient Algorithms*, pages 237–252, Berlin, Germany, 2003. Springer.

[198] M. Soto, A. Rossi, and M. Sevaux. Three new upper bounds on the chromatic number. *Discrete Applied Mathematics*, 159(18) :2281–2289, 2011.

[199] M. Soto, A. Rossi, and M. Sevaux. A mathematical model and a metaheuristic approach for a memory allocation problem. *Journal of Heuristics*, 18(1) :149–167, 2012.

[200] M. Soto, A. Rossi, M. Sevaux, and J. Laurent. *Memory allocation problems in embedded systems*. ISTE Ltd., Croydon, Surrey, UK, 2013.

[201] M Soto. *Optimization methods for the memory allocation problems in embedded systems*. PhD thesis, Université de Bretagne Sud, Lorient, France, 2011.

[202] Tassin Srivarapongse and Phajongjit Pijitbanjong. Solving a special case of the generalized assignment problem using the modified differential evolution algorithms : a case study in sugarcane harvesting. *Journal of Open Innovation : Technology, Market, and Complexity*, 5(1) :5, 2019.

[203] Statista.com. Internet of things - number of connected devices worldwide 2015-2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2016. Accessed : 2021-03-15.

[204] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 409–409, Washington, DC, USA, 2002. IEEE Computer Society.

[205] Larry Stockmeyer. Classifying the computational complexity of problems. *The journal of symbolic logic*, 52(1) :1–43, 1987.

[206] M Sulaiman and A Salhi. A seed-based plant propagation algorithm : the feeding station model. *The Scientific World Journal*, 2015, 2015.

[207] M Sulaiman and A Salhi. A hybridisation of runner-based and seed-based plant propagation algorithms. In *Nature-inspired computation in engineering*, pages 195–215. Springer, 2016.

[208] M Sulaiman, A Salhi, B I Selamoglu, and O B Kirikchi. A plant propagation algorithm for constrained engineering optimisation problems. *Mathematical Problems in Engineering*, 2014 :10 pages, 2014.

[209] M Sulaiman, A Salhi, E S Fraga, Mashwani W.K, and M.M Rashidi. A novel plant propagation algorithm : Modifications and implementation. *Science International*, 28(1) :201–209, 2016.

[210] Daniel Tabak, Albert A Schy, Daniel P Giesy, and KG Johnson. Application of multiobjective optimization in aircraft control systems design. *Automatica*, 15(5) :595–600, 1979.

[211] El-ghazali Talbi. Métaheuristiques pour l'optimisation combinatoire multi-objectif : Etat de l'art. *Rapport CNET (France Telecom) Octobre*, 1999.

[212] L Tang and X Wang. Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem. *The International Journal of Advanced Manufacturing Technology*, 29(11) :1246–1258, 2006.

[213] Jürgen Teich. Pareto-front exploration with uncertain objectives. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 314–328. Springer, 2001.

[214] Paolo Toth and Daniel Vigo. The vehicle routing problem (siam, philadelphia). *Monographs on Discrete Mathematics and Applications, Philadelphia*, 2002.

[215] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4) :333–346, 2003.

[216] S. Udayakumaran and R. Barua. Compiler-decided dynamic memory allocation for scratch-pad based embedded systems. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pages 276–286, New York, NY, USA, 2003. ACM.

[217] S. Udayakumaran, A. Dominguez, and R. Barua. Dynamic allocation for scratch-pad memory using compile-time decisions. *ACM Transactions on Embedded Computing Systems*, 5(2) :472–511, 2006.

[218] Janna Magrietje van den Akker. *LP-based solution methods for single-machine scheduling problems*. PhD thesis, Citeseer, 1994.

[219] J. van Leeuwen. *Handbook of Theoretical Computer Science*. Handbook of theoretical computer science / ed. by Jan. van Leeuwen. Elsevier, 1990.

[220] M. Verma, S. Steinke, and P. Marwedel. Data partitioning for maximal scratchpad usage. In *Proceedings of the Asia and South Pacific Design Automation Conference*, ASP-DAC'03, pages 77–83, 2003.

[221] M. Verma, L. Wehmeyer, and P. Marwedel. Dynamic overlay of scratchpad memory for energy minimization. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 104–109, New York, NY, USA, 2004. ACM.

[222] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of things-global technological and societal trends*, 1(2011) :9–52, 2011.

[223] Xu Wang, MengChu Zhou, Qiuhong Zhao, Shixin Liu, Xiwang Guo, and Liang Qi. A branch and price algorithm for crane assignment and scheduling in slab yard. *IEEE Transactions on Automation Science and Engineering*, 2020.

[224] J M Wilson. A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48(8) :804–809, 1997.

[225] A J Woodcock and J M Wilson. A hybrid tabu search/branch & bound approach to solving the generalized assignment problem. *European Journal of Operational Research*, 207(2) :566–578, 2010.

[226] M Yagiura, T Ibaraki, and F Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2) :133–151, 2004.

[227] M Yagiura, S Iwasaki, T Ibaraki, and F Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization*, 1(1) :87–98, 2004.

[228] M Yagiura, T Ibaraki, and F Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2) :548–569, 2006.

[229] Mehmet Erkan Yüksel. Design and implementation of a batteryless wireless embedded system for iot applications. *Electrica*, 19(1) :1–11, 2019.

[230] Lofti Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE transactions on Automatic Control*, 8(1) :59–60, 1963.

[231] Zizhen Zhang, MengChu Zhou, and Jiahai Wang. Construction-based optimization approaches to airline crew rostering problem. *IEEE Transactions on Automation Science and Engineering*, 17(3) :1399–1409, 2019.

[232] Y Zhou, Y Wang, X Chen, L Zhang, and K Wu. A novel path planning algorithm based on plant growth mechanism. *Soft Computing*, 21(2) :435–445, 2016.

[233] Haibin Zhu. Avoiding conflicts by group role assignment. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 46(4) :535–547, 2015.

[234] Haibin Zhu. Group multi-role assignment with conflicting roles and agents. *IEEE/-CAA Journal of Automatica Sinica*, 7(6) :1498–1510, 2020.