République Algérienne démocratique et populaire
Ministère de l'enseignement supérieure et de la recherche scientifique
Université Mohamed Khider – BISKRA
Faculté des sciences exactes et sciences de la nature et de la vie
Département d'informatique

N° d'ordre :SIOD04 /M2/2023

Mémoire

Présentée en vue de l'obtention du diplôme de Master en

Informatique

Parcours : Systèmes d'information Optimisation et Décision (SIOD)

Réalisation d'une méthode d'optimisation approchée pour la résolution d'un problème NP complet

Par :

CHABIRA INES

Soutenu le 21/06/2023 devant le jury composé de :

Nom Hattab dalila	grade	Président
Kalfali Toufik	MAA	Encadreur
Nom Touil keltoum	grade	Examinateur

Année universitaire 2022-2023

Remerciements

Je reconnais et remercie Dieu, le Tout-Puissant, pour Sa bonté, Son aide et Sa miséricorde qui m'ont permis d'avoir mes prières exaucées. Je tiens à exprimer ma gratitude envers toutes les personnes qui ont consacré leur temps, leur patience et leur savoir pour me fournir une aide inestimable. Sans leur soutien, vous ne tiendriez pas entre vos mains cette modeste tâche.

Je tiens à exprimer ma sincère gratitude envers mon superviseur, l'honorable Mr. T. Kalfali, pour ses précieux conseils et orientations tout au long de la rédaction de ce mémoire.

Je tiens à exprimer ma sincère gratitude aux membres du jury d'avoir accepté d'examiner ce travail. **Ines Chabira**

Dédicace

Dédicace

A

Je dédie ce modeste effort, à mes très chers parents que Dieu les garde. soeurs et frères :Latifa, Houda, Mouna,Amine,Abd el djallele et Sara.

Les enfants de mes sœurs et frères : Sadjed, Mounsef, Yasmine, Rayhana, Loay, Malak, Taha, Iyad, Abdmalek, Emar, Aya et Ayoub.

Mes petites soeurs : Hadjer,Kawther,Maria,rayane,nour.

et mes amis : imane,hayett ,wiam,asma,joujou.

Ines Chabira

Résumé

Dans ce mémoire, nous avons exploré l'importance de l'optimisation combinatoire dans la résolution de problèmes pratiques, en particulier dans le domaine de l'affectation de produits aux machines. Nous avons mis en évidence les différentes méthodes de résolution étudiées, telles que les méthodes exactes, les méthodes heuristiques et les méthodes méta-heuristiques, en soulignant leur applicabilité à différents types de problèmes.

Pour enrichir davantage notre analyse, nous proposons d'ajouter l'algorithme de recuit simulé, une technique de recherche locale méta-heuristique. L'algorithme de recuit simulé est inspiré du processus de recuit utilisé en métallurgie, où un matériau est chauffé puis progressivement refroidi pour obtenir une structure cristalline de meilleure qualité. Dans le contexte de l'optimisation combinatoire, l'algorithme de recuit simulé explore l'espace des solutions en effectuant des mouvements de voisinage et en acceptant des solutions de moins bonne qualité avec une certaine probabilité, ce qui permet d'éviter les minima locaux.

En ajoutant l'algorithme de recuit simulé à notre étude, nous pourrons comparer ses performances avec les autres méthodes déjà mentionnées. Nous pourrons discuter de ses avantages, notamment sa capacité à échapper aux minima locaux, et de ses limites, telles que sa sensibilité aux paramètres de refroidissement et aux choix de voisinage. De plus, nous pourrons examiner comment l'algorithme de recuit simulé s'applique spécifiquement à l'affectation de produits aux machines, en évaluant son efficacité et sa rapidité dans ce domaine.

Mots clés : Optimisation combinatoire, Affectation de produits aux machines, Méthodes de résolution, Méthodes méta-heuristiques,Algorithme de recuit simulé,Minima locaux,Efficacité et rapidité.

ABSTRACT

In this thesis, we have explored the importance of combinatorial optimization in solving practical problems, particularly in the field of product-to-machine assignment. We have highlighted different studied resolution methods, such as exact methods, heuristic methods, and metaheuristic methods, emphasizing their applicability to various types of problems.

To further enhance our analysis, we propose adding the simulated annealing algorithm, a metaheuristic local search technique. The simulated annealing algorithm is inspired by the annealing process used in metallurgy, where a material is heated and gradually cooled to achieve a higher-quality crystalline structure. In the context of combinatorial optimization, the simulated annealing algorithm explores the solution space by making neighboring moves and accepting lower-quality solutions with a certain probability, thus avoiding local minima.

By incorporating the simulated annealing algorithm into our study, we can compare its performance with the other mentioned methods. We can discuss its advantages, such as its ability to escape local minima, and its limitations, such as sensitivity to cooling parameters and neighborhood choices. Additionally, we can examine how the simulated annealing algorithm specifically applies to product-to-machine assignment, evaluating its effectiveness and efficiency in this domain.

Keywords : Combinatorial optimization, Product-to-machine assignment, Resolution methods, Metaheuristic methods, Simulated annealing algorithm, Local minima, Efficiency and speed.

Sommaire

Table des figures

Introduction générale

Le problème d'affectation de produits aux machines est un défi complexe qui se pose dans de nombreux domaines, tels que la logistique, la planification de la production et l'optimisation des ressources.

Son objectif est d'optimiser l'assignation des produits aux machines afin de minimiser les coûts, maximiser l'efficacité et respecter les contraintes spécifiques du système.

Dans cette étude, nous nous intéressons à l'utilisation de l'algorithme de recuit simulé pour résoudre ce problème d'affectation. L'algorithme de recuit simulé est une approche heuristique puissante qui permet d'explorer l'espace des solutions de manière itérative et probabiliste. Il est inspiré du processus de recuit en métallurgie, où un matériau est chauffé puis refroidi lentement pour atteindre un état d'équilibre optimal.

Nous examinons différents aspects clés de l'application de l'algorithme de recuit simulé pour résoudre le problème d'affectation de produits aux machines. Tout d'abord, nous abordons l'affectation des coûts, qui consiste à attribuer des valeurs de coût à chaque affectation produit-machine. Ensuite, nous explorons la génération des solutions initiales, qui peut se faire de manière aléatoire. Enfin, nous nous intéressons aux critères d'arrêt de l'algorithme, qui déterminent quand la recherche doit se terminer.

Cette étude vise à fournir une compréhension approfondie de l'application de l'algorithme de recuit simulé pour résoudre le problème d'affectation de produits aux machines. Nous examinerons des cas d'étude concrets, analyserons les résultats obtenus et discuterons des avantages, des limites et des perspectives d'amélioration de cette approche.

Notre problématique

Dans cette étude est de déterminer comment appliquer efficacement l'algorithme de recuit simulé pour résoudre le problème d'affectation de produits aux machines. Plus

spécifiquement, nous nous intéressons aux aspects suivants :

- **Affectation des coûts :** Comment attribuer les coûts d'affectation aux différentes combinaisons produit-machine de manière à optimiser la solution finale?
- **Génération des solutions initiales :** Comment générer des solutions initiales de haute qualité pour l'algorithme de recuit simulé?
- **Critères d'arrêt :** Quels critères d'arrêt devons-nous utiliser pour déterminer quand l'algorithme de recuit simulé doit s'arrêter ?

En abordant ces questions, nous chercherons à améliorer la qualité des solutions obtenues, la convergence de l'algorithme et à identifier les bonnes pratiques pour l'application de l'algorithme de recuit simulé au problème d'affectation de produits aux machines. Nous utiliserons des études de cas et des expérimentations pour évaluer la performance de l'algorithme et comparer différentes approches.

L'objectif

Dans cette étude est d'appliquer efficacement l'algorithme de recuit simulé pour résoudre le problème d'affectation de produits aux machines. Nous cherchons à atteindre les objectifs suivants :

1. Optimisation de l'affectation des produits aux machines.
2. Exploration efficace de l'espace des solutions.
3. Amélioration de la performance de l'algorithme.
4. Évaluation et comparaison des résultats.

Structure du mémoire

Organisation du mémoire De ce fait, notre mémoire se compose de trois chapitres.

- **Chapitre 1 État de l'art :**

  Ce chapitre présente une introduction à l'optimisation combinatoire et examine les problèmes associés à ce domaine. Il explore les méthodes de résolution exactes et les méthodes de résolution approchées pour les problèmes d'optimisation combinatoire. L'algorithme de recuit simulé est étudié en détail, y compris son historique, son principe de fonctionnement et son application aux problèmes d'optimisation combinatoire.

  Ce chapitre établit les bases essentielles pour comprendre et appliquer l'algorithme de recuit simulé dans le contexte de l'optimisation combinatoire.
- **Chapitre 2 Conception :**

  Dans ce chapitre, nous commencerons par présenter notre problème d'affectation de produits aux machines.Ensuite, nous décrirons la structure globale de notre système, en expliquant comment les différents composants interagissent entre eux. Nous approfondirons ensuite les détails de la conception du système, et en décrivant les choix de modélisation.

Ce chapitre nous permettra de comprendre la conception globale de notre système d'affectation de produits aux machines et de visualiser en détail son fonctionnement.

- **Chapitre 3 Implémentation :**

Dans ce chapitre, nous commencerons par discuter du choix des outils de développement utilisés pour la réalisation de notre système d'affectation de produits aux machines.Ensuite, nous présenterons en détail l'implémentation du système, en décrivant les différentes étapes de développement, les structures de données utilisées, ainsi que les fonctionnalités implémentées.

Ce chapitre nous permettra de comprendre comment le système a été concrètement mis en œuvre, en mettant en évidence les aspects techniques et les décisions prises lors de l'implémentation.

Chapitre 1

Etat de l’art

1.1 Introduction

L'optimisation est le processus de recherche de la meilleure solution possible à un problème donné. Cela implique la maximisation ou la minimisation d'une fonction objective en fonction des valeurs de certaines variables de décision.L'optimisation peut être divisée en deux catégories principales : l'optimisation continue et l'optimisation combinatoire.

L'optimisation combinatoire concerne les problèmes qui ont un nombre fini de solutions possibles. Ce type de problème est souvent rencontré dans des domaines tels que la logistique, la planification, la production, la finance et l'informatique, entre autres.

Le but de ce chapitre est de faire un aperçu sur l'optimisation combinatoire. En commençant par la définition de l'optimisation combinatoire. Ensuite le problème de l'optimisation avec quelques exemples, après ca on va citer quelques méthodes de résolution de ces problèmes.

1.2 Définition d'optimisation

L'optimisation est une discipline qui relève à la fois des mathématiques et de l'informatique. Elle vise à modéliser, analyser et résoudre analytiquement ou numériquement des problèmes visant à déterminer une ou plusieurs solutions satisfaisant un objectif quantitatif tout en respectant d'éventuelles contraintes.

L'optimisation joue un rôle crucial dans divers domaines tels que la recherche opérationnelle, qui se situe à la frontière entre l'informatique, les mathématiques et l'économie. Elle est également essentielle dans les mathématiques appliquées, où elle est fondamentale pour l'industrie et l'ingénierie. L'optimisation trouve également des applications en analyse et en analyse numérique, en statistique pour l'estimation du maximum de vraisemblance d'une distribution, dans la recherche de stratégies dans le cadre de la théorie des jeux, ainsi que dans la théorie du contrôle et de la commande [9].

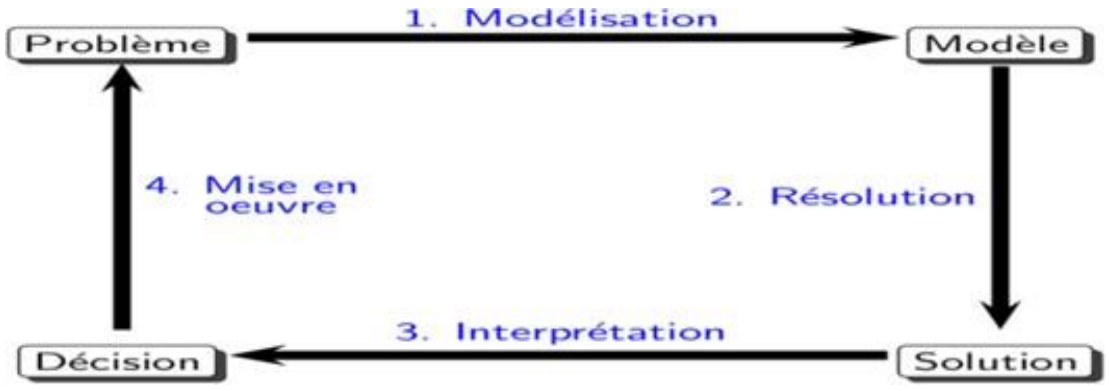


FIGURE 1.1 – Le processus d'optimisation [1]

1.3 Problème d'optimisation

Un problème d'optimisation est un modèle mathématique formel d'un problème réel dans lequel on cherche à minimiser ou maximiser une fonction objectif, tout en respectant des contraintes spécifiques :

$x \in D$: **Ensemble des variables**
$f : \mathbb{R}^n \rightarrow \mathbb{R}$: ***Fonction coût (objectif)***
$D \subset \mathbb{R}^n$: ***Ensemble des contraintes g(x) (Espace se Recherche)***

$$\boldsymbol{PO} = \begin{cases} min\ (ou\ max) f(x) \\ g(x) \leq 0 \\ h(x) = 0 \\ x \in \mathbb{R}^n \end{cases} /$$

FIGURE 1.2 – Formulation mathématique d'un problème d'optimisation

Les problèmes d'optimisation sont divisés naturellement en deux catégories : ceux avec des variables continues, et ceux avec des variables discrètes, qui sont appelés souvent problèmes d'optimisation combinatoire. Dans l'optimisation continue on considère l'ensemble de solutions réalisables comme un ensemble réel, tandis qu'en optimisation combinatoire on cherche un certain objectif dans des ensembles finis ou infinis dénombrables [9].

1.4 Problème d'optimisation combinatoire

Un problème d'optimisation se définit comme la recherche du minimum ou de maximum d'une fonction donnée, mathématiquement, dans le cas d'une minimisation dans le cas d'une minimisation, un problème d'optimisation se présentera sous la forme suivante[17] :

$$\boldsymbol{PO} = \begin{cases} min\ (ou\ max) f(x) \\ g_i(x) \leq 0\ ,\ i = 1 \ldots k \\ h_j(x) = 0\ ,\ j = k+1 \ldots m \\ x \in \mathbb{R}^n \end{cases}$$

FIGURE 1.3 – Minimisation d'un problème d'optimisation[1]

Un problème d'optimisation combinatoire (POC) est un PO = (S ; F) où :

- L'ensemble S des configurations est fini et dénombrable.
- La taille de l'ensemble F croît exponentiellement avec la taille de l'instance du problème.

Les problèmes d'optimisation combinatoire sont souvent de grande taille et leur taille augmente continuellement en raison de l'explosion combinatoire. L'énumération de toutes les solutions n'est pas envisageable. Ces problèmes sont généralement associés à des problèmes de décision, tels que la recherche d'un sous-ensemble optimal, l'affectation de ressources, la planification de production, le routage de véhicules, etc.

La recherche de la solution optimale à un problème d'optimisation combinatoire nécessite un temps de calcul considérable, en particulier lorsque le problème est de grande taille. Les délais de résolution exigés sont de plus en plus courts, ce qui ajoute une contrainte supplémentaire à la résolution de ces problèmes [5].

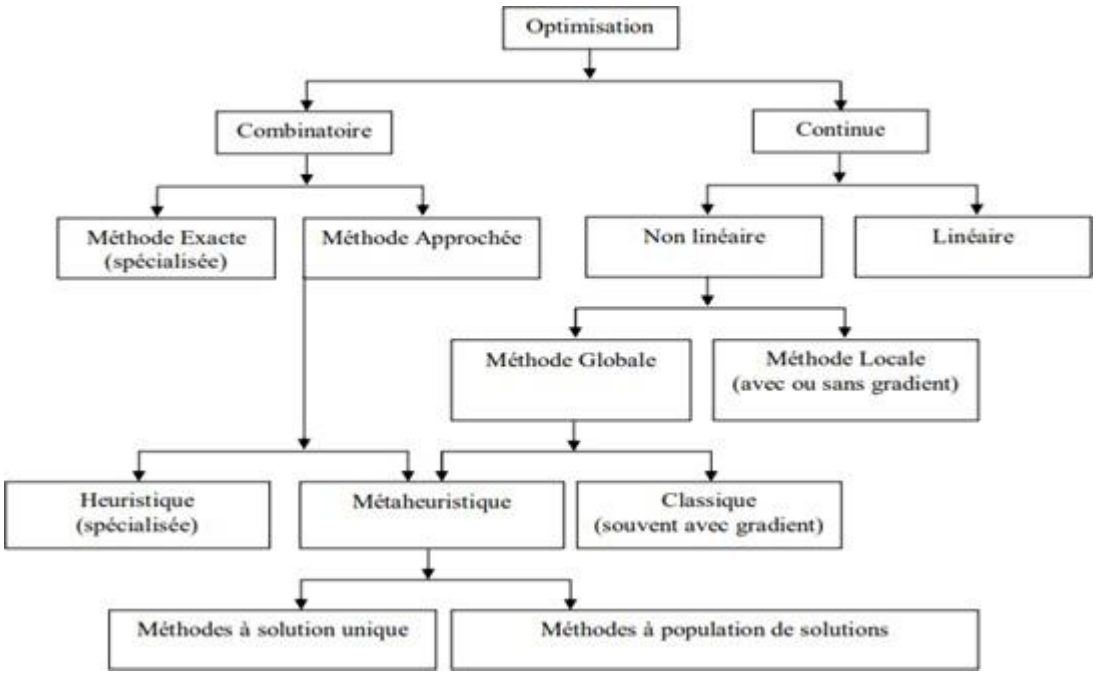


FIGURE 1.4 – Classification générale des méthodes d'optimisation[1]

1.5 Exemples des problèmes d'optimisation combinatoire

Voici quelques exemples de problèmes d'optimisation combinatoire :

1.5.1 Le problème du sac à dos (KP)

Le problème du sac à dos, également connu sous le nom de problème du sac à dos (en anglais, Knapsack Problem), est un exemple de problème d'optimisation combinatoire. Il représente une situation similaire au remplissage d'un sac à dos avec un poids maximal, en choisissant parmi un ensemble d'objets ayant chacun un poids et une valeur. L'objectif est de sélectionner les objets à mettre dans le sac à dos de manière à maximiser la valeur totale, tout en respectant la contrainte de poids maximal [18].

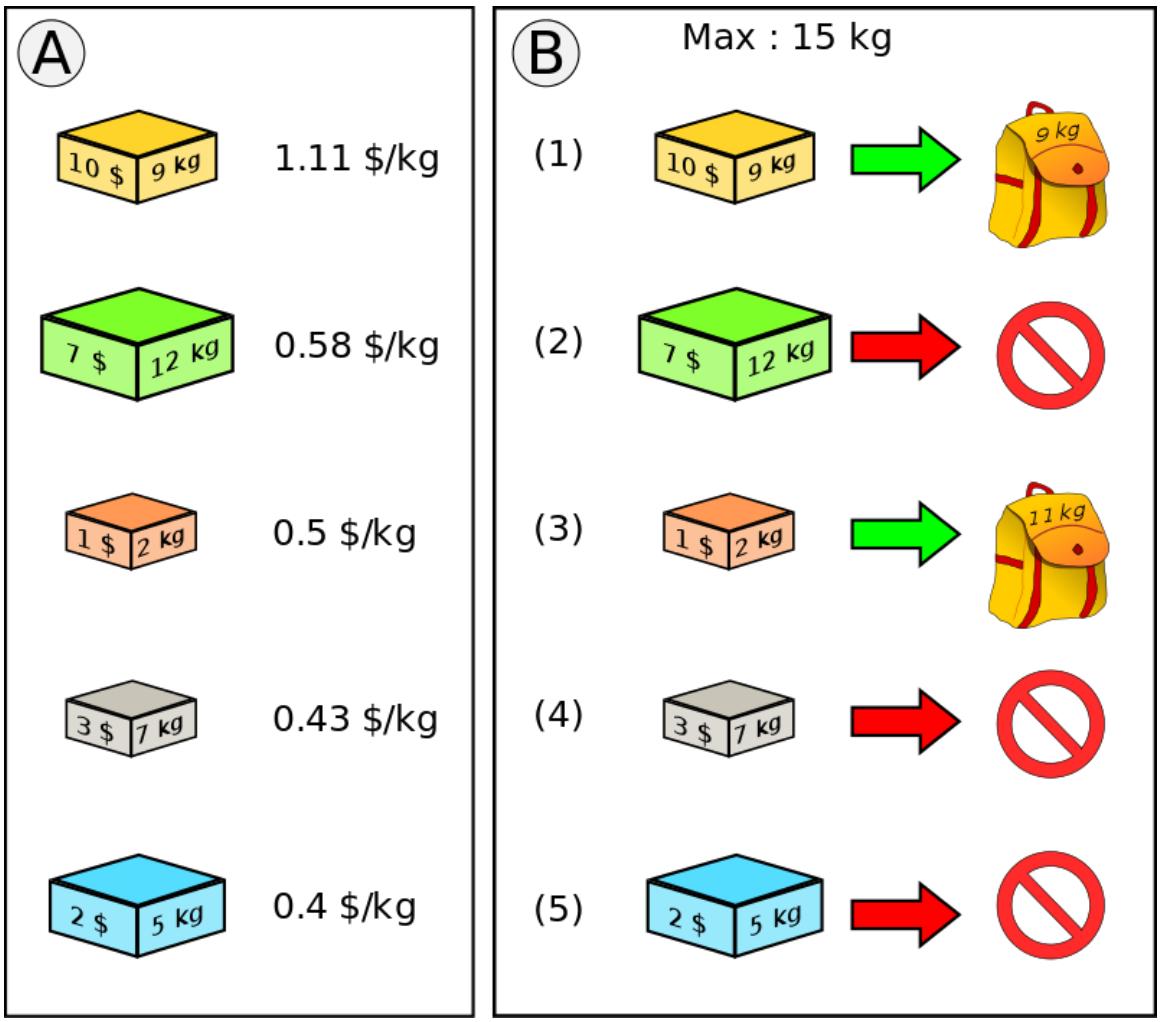


FIGURE 1.5 – Le problème du sac à dos[2]

1.5.2 probleme de voyage de commerce (TSP)

Le "problème du voyageur de commerce", également connu sous le nom de TSP (pour Traveling Salesman Problem), est formulé de la manière suivante : un représentant de commerce doit visiter un ensemble de n villes et souhaite établir un itinéraire qui passe exactement une fois par chaque ville et revient à son point de départ, en parcourant la plus courte distance possible.

Le TSP est l'un des problèmes les plus anciens et les plus étudiés en optimisation combinatoire. Il a de nombreuses applications. Par exemple, des problèmes de sé-

quencement de processus de fabrication ou d'optimisation de parcours en robotique peuvent être formulés directement sous la forme d'un TSP. Certains problèmes, tels que les problèmes de transport, sont plus complexes que le TSP, mais présentent une structure sous-jacente similaire à celle du TSP [19] .

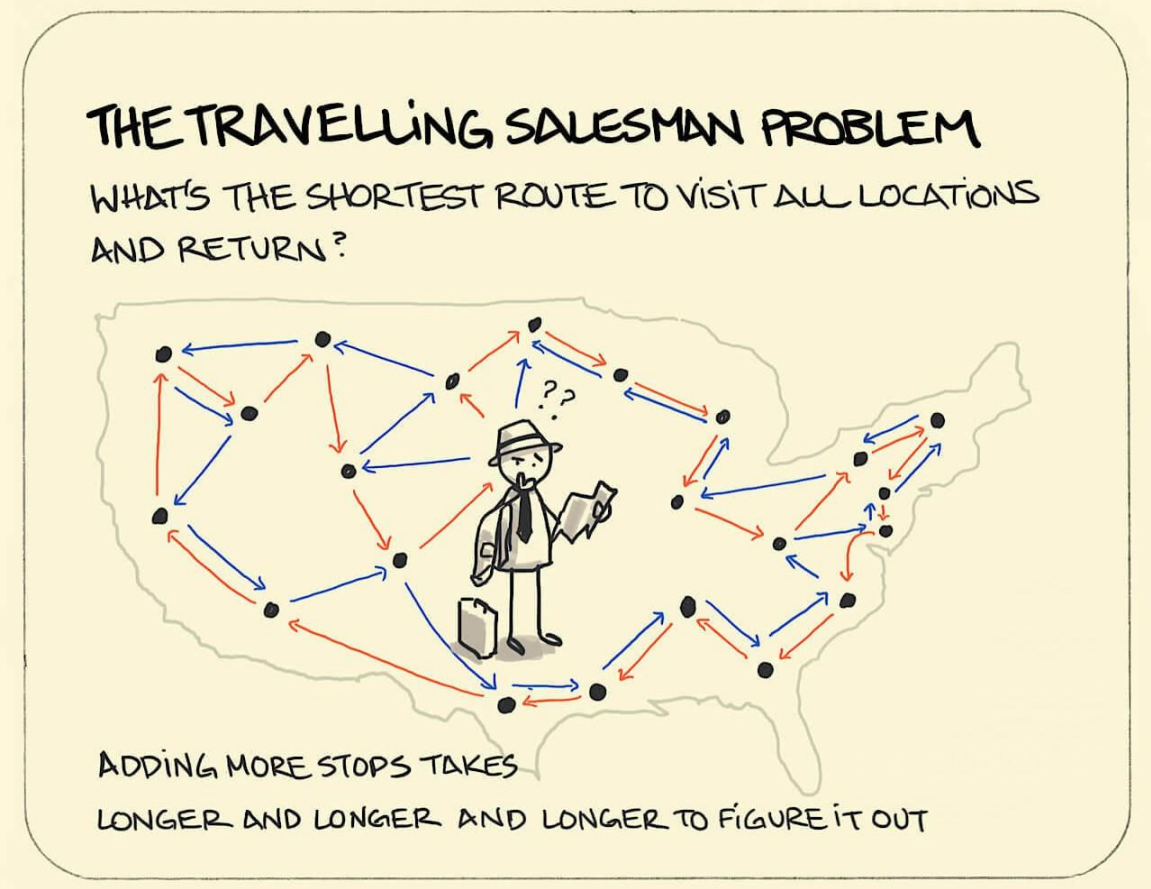


FIGURE 1.6 – Problème du voyageur de commerce[3]

1.5.3 Problème d'affectation

Le "problème d'affectation" est un problème qui vise à établir des correspondances entre les éléments de deux ensembles distincts, tout en minimisant un coût et en respectant des contraintes d'unicité de liaison pour chaque élément.Considérons m tâches et n agents, avec $n \geq m$. Pour chaque couple (i, j) (où i varie de 1 à m et j varie de 1 à n), l'affectation de la tâche i à l'agent j entraîne un coût de réalisation noté $c_{i,j}$ (où $c_{i,j} \geq 0$). Chaque tâche doit être réalisée exactement une fois et chaque agent peut réaliser au plus une tâche.

Le problème consiste à attribuer les tâches aux agents de manière à minimiser le coût total de réalisation, tout en respectant les contraintes de réalisation des tâches et la disponibilité des agents [19].

1.5.4 Problème d'ordonnancement

L'ordonnancement, comme son nom l'indique, consiste à organiser les différentes tâches d'un projet dans le temps, avec pour objectif d'optimiser la planification de ces tâches. C'est une technique de gestion visant à respecter les délais établis tout en minimisant les coûts.Plus précisément, l'ordonnancement est le processus par lequel des priorités successives sont attribuées à des tâches différentes. C'est l'ordonnanceur qui est chargé de cette activité.

L'ordonnancement implique l'organisation temporelle de l'exécution d'une séquence de tâches, en prenant en compte diverses contraintes de production :

- Contraintes temporelles : délais requis, respect des échéances, priorités.
- Contraintes techniques : contraintes d'enchaînement des tâches, spécificités technologiques des machines ou équipements utilisés.
- Contraintes de capacité : disponibilité des ressources nécessaires à l'exécution des tâches, telles que la main-d'œuvre, les équipements, les matériaux, etc.

L'objectif de l'ordonnancement est de planifier de manière efficace et optimale les différentes tâches d'un projet, en prenant en compte ces contraintes, afin d'assurer une exécution fluide et rentable [1].

1.5.5 Problème de tournée de véhicule (VRP)

Problème du VRP : Le problème du véhicule routier (VRP) consiste à trouver une partition de m routes, notées R_1, R_2, . . . , R_m, de l'ensemble des villes (V) de manière à minimiser le coût total, tout en respectant les contraintes suivantes :

- Chaque route Ri doit commencer et se terminer au dépôt, noté v0.
- Chaque client doit être visité une seule fois par un seul véhicule.
- La demande totale d'une route ne doit pas dépasser la capacité Q du véhicule.
- La durée d'une route ne doit pas dépasser une certaine borne D.
- Le VRP est une généralisation du problème du voyageur de commerce (TSP) [5].

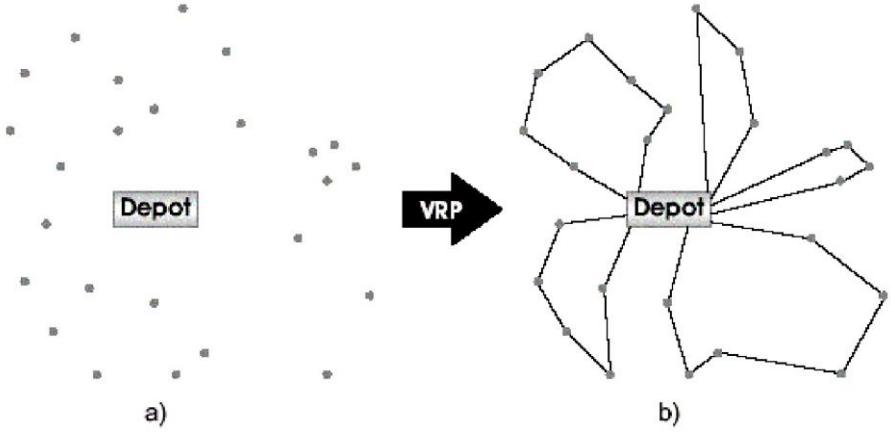


FIGURE 1.7 – Problème de tournée de véhicule [4]

1.5.6 Problème de docking moléculaire (MolecularDocking)

Le docking moléculaire est une problématique qui consiste à déterminer la disposition relative de deux molécules, à savoir un ligand et un récepteur. La structure résultante du docking confère des propriétés spécifiques à l'ensemble formé, appelé complexe. La recherche de la structure optimale est d'une importance capitale dans le processus de conception de nouveaux médicaments [4].

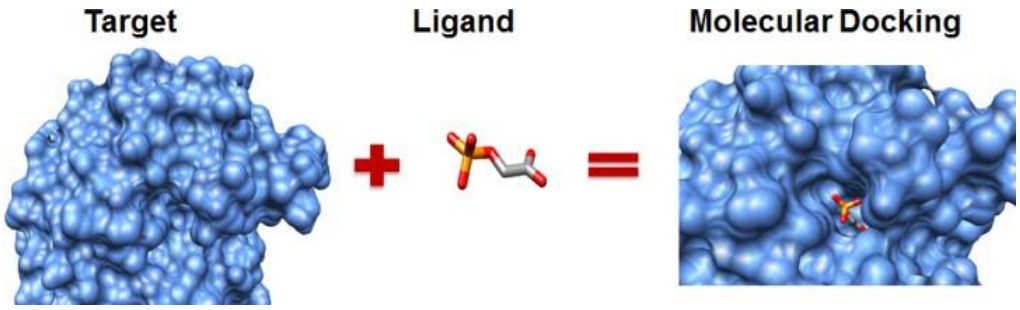


FIGURE 1.8 – Le docking moléculaire[5]

1.5.7 Le problème d'affectation des produits aux machines(APM)

Le problème d'affectation des produits aux machines (APM) consiste à résoudre l'allocation optimale des produits dans un atelier composé de m machines pour fabriquer n produits. Chaque produit peut être entièrement fabriqué par une machine spécifique, ou il peut ne pas être fabriqué du tout. Chaque paire produit-machine a un temps de fabrication associé, indiquant le temps nécessaire pour fabriquer ce produit sur cette machine particulière. De plus, chaque machine a une limite de temps de travail total.

L'objectif est donc de trouver une allocation des produits aux machines qui minimise le coût total. Cette allocation optimale permet d'optimiser l'utilisation des ressources de l'atelier en répartissant les produits de manière efficace entre les machines. Ainsi, le temps de travail total est minimisé, ce qui peut conduire à une meilleure productivité et à une réduction des coûts de fabrication [20].

1. Les variables de décision : $\{(i = 1..m, j = 1..n)\}$ avec i représentant la machine et j représentant le produit.
2. L'expression mathématique des contraintes du problème : Chaque machine i est caractérisée par un temps total de travail noté T_i. Le temps pris par la machine i pour fabriquer le produit j est noté T_{ij}.
   - Chaque produit j est fabriqué par une et une seule machine i : $\sum T_{ij} = 1$ pour tout j.
   - Chaque machine i peut fabriquer plusieurs produits j en respectant la contrainte suivante : $\sum T_{ij} \leq T_i$ pour tout i.
3. L'expression mathématique de la fonction objectif à optimiser : On cherche donc à déterminer une affectation des produits aux machines qui minimise le coût total. Coût total = $\sum \sum C_{ij} \cdot T_{ij}$ où C_{ij} représente le coût de fabrication du produit j par la machine i.

Le APM est alors modélisé comme suit [20] :

$$Min(\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij})$$

$\sum_{i=1}^{n} C_{ij} = 1, i = 1, n$
$\sum_{j=1}^{n} C_{ij} = 1, j = 1, m$

1.6 Domaines d'application

L'optimisation combinatoire intervient sur toute situation nécessitant une planification ou une organisation efficace. Elle est liée à la minimisation de risque, du temps ou du coût ainsi qu'à la maximisation de la qualité, du profit ou de l'efficacité. L'industrie et l'ingénierie sont les domaines qui profitent le plus des études de l'optimisation combinatoire. On cite quelques exemples :

- La conception de réseaux (électriques, téléphoniques, etc.).
- La planification des tâches des employés d'une entreprise.
- Le calcul d'itinéraire pour les applications de géolocalisation.
- La planification du trafic aérien ou routier.
- La gestion de portefeuilles.
- L'allocation de ressources matérielles et humaines.
- L'ordonnancement et la planification de la production.
- La maximisation du profit et la minimisation du coût [21].

1.7 Méthodes de résolution d'un problème d'optimisation

La résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées.

Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales :

- La classe de méthodes exactes et la classe de méthodes approchées.
- L'hybridation des méthodes de ces deux classes a donné naissance à une pseudo classe qui englobe des méthodes dites hybrides [8].

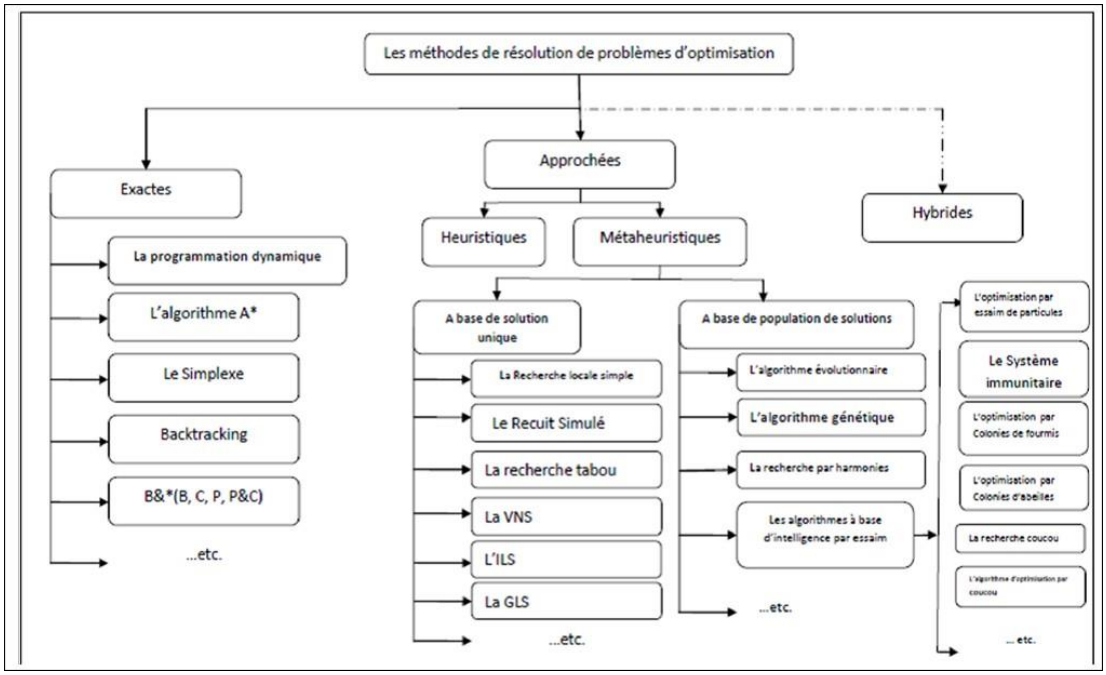


FIGURE 1.9 – Classification des méthodes d'optimisation [1]

1.7.1 Méthodes exactes

Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire. C'est la raison pour laquelle, elles sont beaucoup plus utilisées pour la résolution de problèmes faciles [9].

1.7.1.1 Branch and Bound

L'algorithme de Branch and Bound (Séparation et évaluation) est une méthode exacte pour la résolution d'un problème d'optimisation. Il se structure comme une énumération « intelligente » de l'espace de recherche ou du domaine admissible du problème.

L'algorithme est basé sur une recherche arborescente, générant un arbre. Comme son nom l'indique, l'algorithme se compose de deux procédures :

- La séparation : séparer l'ensemble de solutions en sous-ensembles.
- L'évaluation : évaluer les solutions d'un sous-ensemble en majorant la valeur de la meilleure solution de ce sous ensemble [1].

1.7.1.2 Programme linéaire

Un Programme Linéaire (PL) est un problème d'optimisation consistant à maximiser (ou minimiser) une fonction objectif linéaire de variables soumises à un ensemble de contraintes exprimées sous forme d'équations ou d'inéquations linéaires.

$$(P_0)\begin{cases} Max(Min)\ z = & \sum_{j=1}^{n} c_j x_j & & & \text{(fonction objectif)} \\ \text{s.c.} & \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i \quad i \in I^- \subseteq \{1,\ldots,m\} & \text{(contraintes inégalité)} \\ & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i \quad i \in I^+ \subseteq \{1,\ldots,m\} & \text{(contraintes inégalité)} \\ & \sum_{j=1}^{n} a_{ij} x_j & = & b_i \quad i \in I^0 \subseteq \{1,\ldots,m\} & \text{(contraintes égalité)} \\ & x_j \geq 0 & & j = 1,\ldots,n & \text{(contraintes de non-négativité)} \end{cases}$$

FIGURE 1.10 – La forme générale d'un programme linéaire

Où les ensembles I^-, I^+ et I^0 sont disjoints, $I = I^- \cup I^+ \cup I^0 = \{1, \ldots, m\}$, et où c_j, a_{ij} et b_i $(i = 1, \ldots, m, j = 1, \ldots, n)$ sont des constantes supposées connues. P_0 est un programme linéaire à n variables et m contraintes [19].

1.7.2 Méthodes approchées

Une méthode approchée est une méthode d'optimisation qui vise à trouver une solution réalisable de la fonction objectif dans un temps raisonnable, mais sans garantie d'optimalité. Ces méthodes sont avantageuses car elles peuvent être appliquées à différentes classes de problèmes, qu'ils soient simples ou très complexes. Par exemple, les algorithmes de recuit simulé, les algorithmes tabous et les algorithmes génétiques ont démontré leur robustesse et leur efficacité dans la résolution de nombreux problèmes d'optimisation combinatoire .Les méthodes approchées peuvent être regroupées en deux catégories principales : les heuristiques et les méta-heuristiques [9].

1.7.2.1 Méthodes heuristiques classique ou constructives

Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier Elles nécessitent des connaissances du domaine du problème traité. En fait, ce sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures.Les heuristiques sont des ensembles de règles empiriques ou des stratégies qui fonctionnent, en effet, comme des règles d'estimation [8].

- **L'Algorithme glouton**

  Les algorithmes gloutons (greedy algorithms) offrent une approche plus simple à mettre en œuvre, mais leur résultat n'est pas toujours optimal, sauf dans certaines situations spécifiques appelées situations canoniques.L'approche gloutonne consiste à construire une solution complète en effectuant une série de choix locaux qui conduisent systématiquement à la meilleure solution partielle.

  En résumé, on peut utiliser un algorithme glouton lorsque les conditions suivantes sont remplies :

  - Une solution complète peut être construite en combinant des solutions partielles successives.
  - Chaque solution partielle est déterminée en effectuant un choix local basé sur la solution partielle précédente.

- On dispose d'une fonction d'évaluation permettant d'estimer la qualité de chaque solution partielle [22].

1.7.3 Méthodes MétaHeuristiques

Le mot méta-heuristique est composé de deux mots grecs : méta et heuristique. Le mot méta est un suffixe signifiant au-delà c'est-à-dire de niveau supérieur. Les méta-heuristiques sont des méthodes généralement inspirées de la nature. Contrairement aux heuristiques, elles s'appliquent à plusieurs problèmes de nature différentes. Pour cela on peut dire qu'elles sont des heuristiques modernes, de plus haut niveau, dédiées particulièrement à la résolution des problèmes d'optimisation.

Leur but est d'atteindre un optimum global tout en échappant les optima locaux, les méta-heuristiques regroupent des méthodes qui peuvent se diviser en deux classes :Méta-heuristiques à population de solutions et Méta-heuristiques à solution unique [1].

1.7.4 Différence entre heuristique et métaheuristique

Une méta-heuristique est généralement une marche aléatoire dans l'espace de recherche guidée par des heuristiques. Le préfixe "méta" indique que ce sont des algorithmes ultra-génériques qui peuvent être appliqués à de nombreuses catégories de problèmes différents.

À l'opposé, une heuristique reste spécifique à un problème donné. Elle représente simplement une méthode de résolution individuelle, une idée pour résoudre un problème spécifique [8].

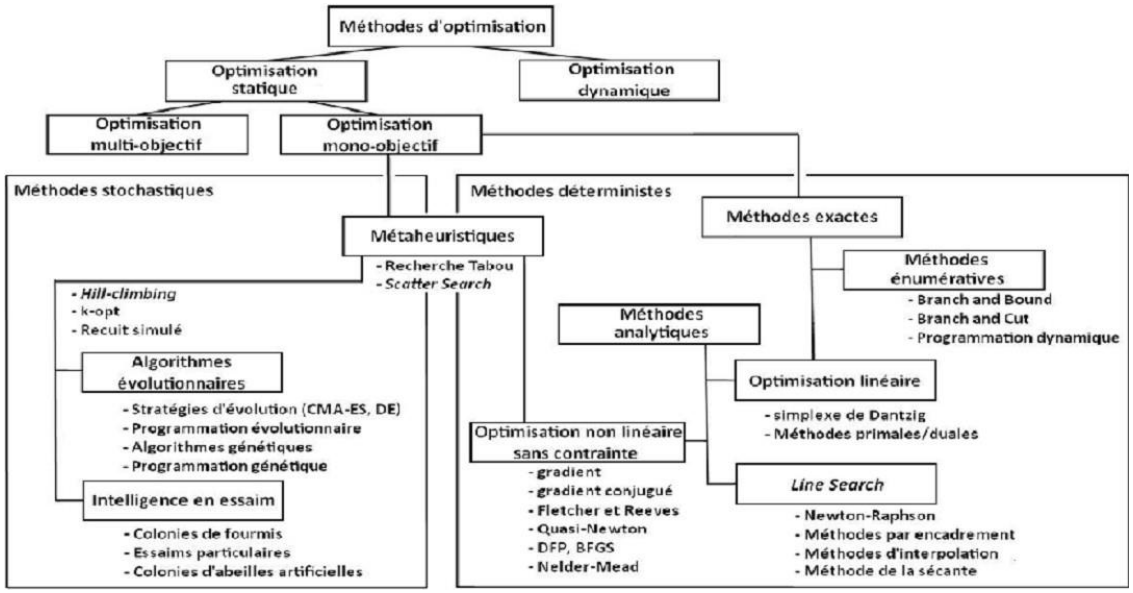


FIGURE 1.11 – Classification des méthodes d'optimisation[1]

1.7.5 Méta-heuristiques à solution unique

Ces méthodes traitent une seule solution à la fois, afin de trouver la solution optimale.

1.7.5.1 Méthode de descente (Hill Climbing)

La méthode de descente est une méthode classique de recherche locale. Son principe général est le suivant :

1. Débuter avec une solution initiale x dont la valeur objectif est $f(x)$.
2. Choisir une solution voisine x_0 de x telle que $f(x_0) < f(x)$, puis remplacer x par x_0 et répéter jusqu'à ce que pour tout voisin x_0 de $x, f(x) \leq f(x_0)$.

Ainsi, x est un optimum local. Pour trouver l'optimum global, une version améliorée de l'algorithme de descente appelée Iterated Local Search (ILS) consiste à changer la direction de recherche lorsqu'un optimum local est trouvé, en repartant d'une nouvelle solution générée aléatoirement [23].

1.7.5.2 Recuit simulé

Le recuit simulé (RS) est une nouvelle approche pour approximer les solutions des problèmes d'optimisation combinatoires. Cette méthode s'est révélée efficace pour traiter les POC s difficiles.

Le recuit simulé est une version probabiliste de l'algorithme d'optimisation locale(l'optimisation locale est un algorithme itératif de recherche de minima dans un voisinage d'une solution initiale donnée).Dans un schéma de recuit, la transition d'une solution vers une solution voisine est aléatoire ce qui permet de retenir éventuellement des solutions de coût supérieur et d'éviter ainsi de stagner dans un minimum local (par opposition à l'optimisation locale où on transite toujours vers les solutions de moindre coût)[24].

1.7.6 Méta-heuristiques à population de solutions

Ces méthodes utilisent une population de solutions à chaque itération jusqu'à l'obtention de la solution globale .

1.7.6.1 Algorithmes génétiques

Les algorithmes génétiques ont été développés dans les années 1970, s'inspirant de la théorie de l'évolution de Darwin. Dans un algorithme génétique, une population d'individus est créée, simulant ainsi une génération. Ces individus se reproduisent en croisant leurs caractéristiques pour produire une descendance combinant les traits de leurs parents. Des mutations aléatoires peuvent également se produire, modifiant ainsi les caractéristiques de la nouvelle génération. Ensuite, une sélection est effectuée, simulant la sélection naturelle, où seuls les individus les mieux adaptés survivent [25].

Pour mettre en place un algorithme génétique, les individus représentent des solutions potentielles au problème étudié, et la sélection se fait en évaluant leur qualité à l'aide d'une fonction objectif. Les croisements entre les individus sont réalisés à l'aide d'opérateurs de croisement spécifiques. Pour en savoir plus sur ces algorithmes, vous pouvez écouter le podcast "Quand des algorithmes s'inspirent de la théorie de l'évolution".

Dans le cas spécifique du problème du voyageur de commerce, la première étape consiste à créer une population initiale, c'est-à-dire un ensemble significatif d'individus, où chaque individu représente une solution potentielle. Pour cela, il est nécessaire de définir un codage pour représenter chaque individu. Il existe deux types de codage : le codage direct, où l'individu représente directement la solution du problème, et le codage indirect, où l'individu représente une manière de construire la solution. Dans notre cas, nous utiliserons le codage direct, où chaque individu est représenté par une liste ordonnée des numéros des villes à visiter.

Cela permet d'avoir une première compréhension de la méthode des algorithmes génétiques et de son application au problème du voyageur de commerce[22].

Pour initialiser une population, on peut lancer plusieurs fois un algorithme glouton, avec des paramètres différents pour obtenir des solutions différentes. Puis, pendant un certain nombre d'itérations, les étapes suivantes sont exécutées :

- Tout d'abord, chaque individu est évalué. Pour cela, on calcule la valeur de la fonction objectif, c'est-à-dire la longueur du cycle parcouru par le voyageur de commerce.
- Puis, une étape de sélection est appliquée. Cette étape permet d'éliminer les moins bons individus et de garder uniquement les meilleurs en fonction de leur évaluation. Il existe plusieurs méthodes de sélection. Une sélection par rang ne fait pas intervenir le hasard, contrairement à la roulette, car elle choisit les n meilleurs individus de la population. Chaque individu a ainsi une probabilité de sélection dépendant de son évaluation, avec une plus forte probabilité pour les meilleurs individus.
- L'étape suivante consiste à croiser les individus précédemment sélectionnés pour obtenir une nouvelle population. Deux parents sont donc choisis pour appliquer un opérateur de croisement afin d'obtenir un descendant (nouvel individu). Il existe de nombreuses techniques de croisement; dans le cas présent, nous utiliserons le « crossover en un point ». Cet opérateur consiste à recopier une partie du parent 1 et une partie du parent 2 pour obtenir un nouvel individu. Le point de séparation des parents est appelé point de croisement. Il faut cependant faire attention à ne pas visiter plusieurs fois la même ville (on ne recopie pas les villes déjà visitées), et à ne pas oublier de ville (on rajoute à la fin les villes non prises en compte). Voici un exemple avec 8 villes et un point de croisement juste après la troisième ville :

Parent 1	Parent 2	Descendant
1	2	1
2	6	2
3	1	3
4	8	8
5	4	4
6	3	5
7	5	7
8	7	6

FIGURE 1.12 – Un exemple avec 8 villes[6]

- Enfin, avant de revenir à la première étape, un procédé de mutation est utilisé pour diversifier les solutions au fur et à mesure des générations. Cette mutation consiste à modifier aléatoirement une petite partie d'un caractère dans certains individus de la nouvelle génération. Cette étape est effectuée avec une très faible probabilité, et consiste par exemple à échanger deux villes consécutives dans un individu.

Dans certains cas, après chaque croisement et mutation, une étape d'optimisation des descendants est appliquée. L'utilisation d'un algorithme de type recherche locale est adapté pour effectuer cette optimisation rapidement. Nous parlerons alors d'algorithme mimétique. Ce type d'algorithme est intéressant pour obtenir plusieurs solutions de bonne qualité. En effet, à la fin de l'algorithme, la population est constituée des meilleures solutions trouvées [6].

1.7.7 L'intelligence en essaim

L'intelligence en essaim désigne le comportement d'insectes sociaux dont les interactions donnent une cohérence au groupe. Malgré leur éloignement du milieu biologique, le terme a été introduit par Beni et Wang pour décrire un ensemble de robots travaillant en coopération afin de résoudre un problème. Bonabeau et ses collaborateurs [Bonabeau et al., 1999] caractérisent l'intelligence en essaim par un ensemble d'individus simples, communicants et formant une population auto-organisée, adaptative et capable de résoudre des problèmes complexes [26].

1.7.7.1 L'optimisation par essaim particulaire

La méthode s'inspire du comportement social des animaux évoluant en essaim, tels que les nuées d'oiseaux et les bancs de poissons. Comme ces animaux se déplacent en groupe pour trouver de la nourriture ou éviter les prédateurs, les algorithmes à essaim de particules cherchent des solutions à des problèmes d'optimisation.

Dans cet algorithme, les individus sont appelés particules et la population est appelée essaim. Chaque particule décide de son prochain mouvement en se basant sur son expérience personnelle, qui correspond à la meilleure position qu'elle a rencontrée jusqu'à présent, ainsi que sur la position de son meilleur voisin[27].

1.7.7.2 Algorithme de Colonie d'Abeilles Artificielles

Cette technique est développée en inspectant le comportement des abeilles réelles pour trouver la source de nourriture (solution possible), qui s'appelle le nectar, et partager l'information des sources de nourriture aux autres abeilles dans le nid. Chaque solution représente une position de nourriture potentielle dans l'espace de recherche et la qualité de la solution correspond à la qualité de la position alimentaire.

Dans cet algorithme, les abeilles artificielles sont définies et classifiées en trois groupes :

- Abeilles employeuses qui recherchent la source de nourriture (solutions possibles) à partir d'un ensemble prédéfini des sources de nourriture et de partager cette information (danse frétillante de communication des abeilles) avec les autres abeilles dans la ruche.
- Abeilles spectatrices (abeilles d'observation) qui en fonction des informations qu'elles prennent des abeilles employeuses, elles recherchent une meilleure source de nourriture dans le voisinage des sources de nourriture mémorisées.
- Abeilles éclaireuses (scouts) qui sont chargées de trouver de nouvelles nourritures (le nectar de nouvelles sources) [7].

1.7.7.3 Les algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis (ACO) ont été initialement utilisés pour trouver des solutions quasi-optimales au problème du voyageur de commerce, ainsi que pour d'autres problèmes d'optimisation combinatoire. Par la suite, leur utilisation s'est étendue à plusieurs domaines.

Dans l'algorithme ACO, les fourmis partent initialement de la source et explorent les différents chemins possibles jusqu'à atteindre leur destination. Au départ, toutes les pistes ont des quantités de phéromones égales. Sur leur chemin de retour, les fourmis déposent de la phéromone sur les différentes pistes, ce qui permet d'influencer les choix des futures fourmis.

Cet échange de phéromones entre les fourmis permet une communication indirecte et collective, qui permet de trouver des solutions de qualité dans des problèmes d'optimisation [27].

Dans la figure suivante, on présente une illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. On peut observer :

a) Recherche sans obstacle.
b) Apparition d'un obstacle.
c) Recherche du chemin optimal.
d) Chemin optimal trouvé.

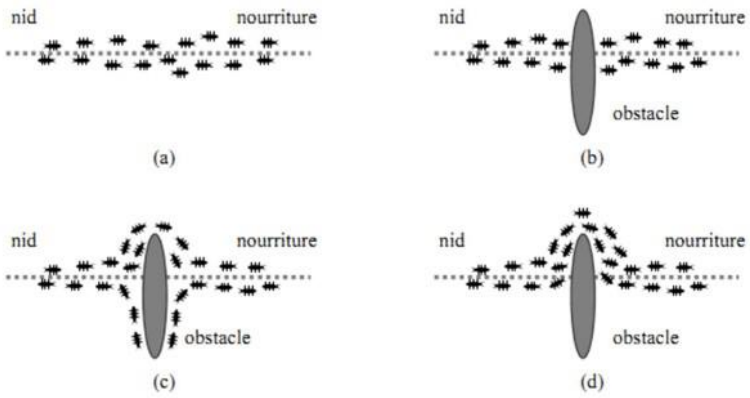


FIGURE 1.13 – Illustration de la recherche optimale de nourriture par les fourmis en minimisant les parcours [7]

1.7.8 Méthodes Hybrides

L'utilisation des méthodes hybrides permet de combiner les avantages des deux types de méthodes (déterministes et non déterministes). Elles peuvent être considérées comme une solution idéale pour surmonter les inconvénients des méthodes locales et des méthodes globales.

En effet, les méthodes locales conduisent à une solution limitée à un voisinage restreint, tandis que les méthodes globales peuvent être très chronophages. Il est donc évident qu'un compromis entre l'exploitation et l'exploration doit être trouvé, d'où l'utilité des méthodes hybrides [28].

1.8 Travaux connexes

Dans le domaine de l'optimisation combinatoire, les travaux connexes sur le problème d'affectation ont attiré l'attention de la communauté scientifique depuis de nombreuses décennies. Au fil du temps, un grand nombre d'approches ont été développées pour résoudre ce problème, en utilisant à la fois des méthodes exactes et des méthodes heuristiques. Ces efforts de recherche démontrent l'importance accordée à la résolution efficace du problème d'affectation, tant dans sa forme exacte que dans des approches plus approximatives.

1.8.1 Le problème d'affectation :

Il existe deux catégories d'algorithmes pour résoudre le problème d'affectation.
La première catégorie comprend les algorithmes exacts tels que la méthode de branch and bound[29], branch and cut[30], l'algorithme Concorde et la programmation dynamique[31]. Ces algorithmes visent à trouver la solution optimale d'affectation. D'autre part.
La deuxième catégorie comprend des algorithmes heuristiques intelligents, qui peuvent être subdivisés en trois sous-catégories : la construction de tours avec des méthodes telles que l'algorithme glouton, l'amélioration de tours avec des algorithmes locaux[32], et les algorithmes composites tels que le recuit simulé[33], l'optimisation par colonie de fourmis[32], l'optimisation par essaim de particules[34], la recherche tabou[35] et les réseaux neuronaux[36].

1.8.2 Le problème d'affectation produits aux machines :

La résolution du problème d'affectation de produits aux machines peut être abordée en utilisant un algorithme génétique. Cet algorithme génétique est une méthode d'optimisation inspirée des processus de l'évolution naturelle. Il utilise des opérations génétiques telles que la sélection, le croisement et la mutation pour explorer l'espace des solutions possibles et trouver une allocation optimale des produits aux machines. En utilisant des techniques de recherche adaptatives et des critères de fitness, l'algorithme génétique peut progressivement améliorer les solutions au fil des générations. Cette approche heuristique offre une alternative efficace pour résoudre le problème d'affectation de produits aux machines en exploitant les principes de l'évolution biologique [37].

1.9 Implantation des Algorithmes

1.9.1 Les problèmes NP-complets

Les problèmes NP-complets ou NP-difficiles d'optimisation sont en général caractérisés par une complexité exponentielle ou factorielle, ce qui signifie que le temps nécessaire pour trouver la solution exacte augmente très rapidement avec la taille du problème. Dans certains cas, cela peut rendre la recherche de la solution exacte impossible en pratique [9].

Les heuristiques et les méta-heuristiques sont des méthodes approximatives qui permettent de résoudre des problèmes d'optimisation difficiles en un temps raisonnable.

1.9.2 Terminologie

Dans le but de bien éclaircir le domaine de notre thème, nous en présentons ici quelques termes que nous jugeons utiles pour la compréhension du travail qui sera abordé plus loin :

1.9.2.1 La Construction

C'est un principe couramment utilisé dans la résolution de problèmes. Elle consiste à construire une solution en procédant par étapes successives, en effectuant à chaque

étape un choix partiel et définitif qui permet d'avancer vers la solution complète [21].

1.9.2.2 La recherche locale

Est une technique d'optimisation qui consiste à partir d'une solution initiale complète et à construire progressivement une suite de solutions de coût de plus en plus faible en explorant les solutions dans un ensemble localement proche, également appelé voisinage. Le choix de la nouvelle solution se fait donc à partir de cet ensemble de solutions voisines [38] .

1.9.2.3 La recherche de voisinage

Consiste à explorer un ensemble de solutions voisines à partir d'une solution initiale donnée pour trouver une solution optimale. Cette technique peut être utilisée pour résoudre des problèmes d'optimisation combinatoire et implique de modifier légèrement la solution courante pour explorer l'espace des solutions voisines. Chaque nouvelle solution est évaluée en fonction d'un critère d'optimisation, et la meilleure solution trouvée est gardée pour être utilisée comme point de départ pour la recherche suivante[1].

1.9.2.4 Espace de recherche et Solution admissible

Représente l'ensemble des valeurs que les variables de décision peuvent prendre pour résoudre un problème donné. Cet espace est souvent noté X et est de dimension n, où n est le nombre de variables de décision dans le problème [21].

1.9.2.5 L'optimum global

Est la solution qui a la meilleure valeur de la fonction objectif parmi toutes les solutions possibles de l'espace de recherche. Il est unique et représente la solution optimale pour le problème d'optimisation donné [9].

1.9.2.6 L'optimum local

Est la meilleure solution dans un voisinage restreint de la solution courante, c'est-à-dire qu'il est localement optimal. Cela signifie que parmi toutes les solutions voisines de la solution courante, l'optimum local a la meilleure valeur de la fonction objectif. Cependant, il est important de noter que l'optimum local ne garantit pas que c'est la solution optimale pour l'ensemble de l'espace de recherche [9].

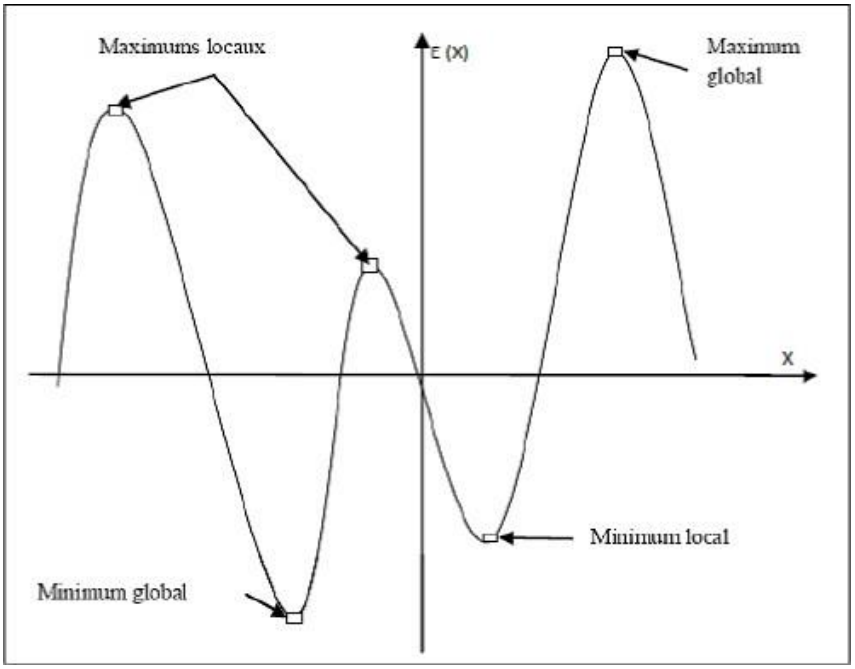


FIGURE 1.14 – Les Optimums d'un Problème d'Optimisation[8]

1.9.3 Recuit Simulé

1.9.3.1 Historique

La méthode du recuit simulé a été introduite par S. Kirkpatrick, C. D. Gelatt Jr. et M. P. Vecchi en 1983 [8].

1.9.3.2 En Métallurgie

Les métaux ont des propriétés uniques, notamment en raison de leurs comportements vis-à-vis des changements de température. L'un de ces comportements est le recuit, où le métal est chauffé puis refroidi lentement au fil du temps. Grâce à ce processus, le métal devient moins dur et plus ductile [9].

1.9.3.3 En Informatique

Les programmeurs informatiques ont pris note de ces propriétés uniques et les ont transposées dans leur domaine, ce qui a donné naissance à l'algorithme du "recuit simulé" (simulated annealing). Il est utilisé pour trouver un optimum global à un problème spécifique, et fonctionne de manière similaire au recuit métallurgique en réduisant progressivement la "température" (qui représente la probabilité de se diriger vers une solution moins "bonne" dans le contexte de l'algorithme) [1].

1.9.3.4 L'algorithme de Metropolis

Dans l'algorithme de Metropolis, on part d'une configuration donnée et on lui applique une modification aléatoire. Si cette modification fait diminuer la fonction objec-

tif (ou l'énergie du système), elle est directement acceptée. Sinon, elle n'est acceptée qu'avec une probabilité égale à $\Delta E = E_{k+1} - E_k = \Delta f = f(x_{k+1}) - f(x_k)$ (où E représente l'énergie, T représente la température et f est notre fonction coût).

Cette règle est connue sous le nom de critère de Metropolis et elle correspond à la différence entre l'énergie précédente et l'énergie actuelle (généralement, l'énergie est représentée par notre fonction coût) [39].

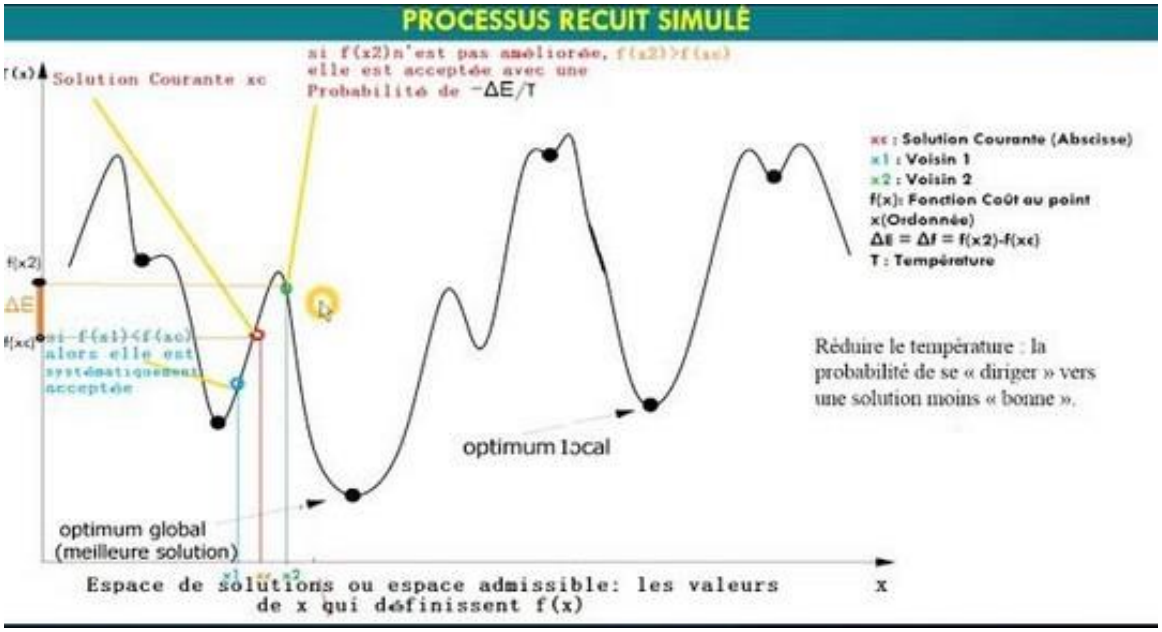


FIGURE 1.15 – Processus Recuit Simulé[9]

1.9.4 Les paramètres de l'algorithme de recuit simulé

Le recuit simulé applique itérativement l'algorithme de Metropolis, pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique :

1. Choisir une température de départ T=T0 et une solution initiale S=S0.
2. Générer une solution aléatoire dans le voisinage de la solution actuelle.
3. Comparer les deux solutions selon le critère de Metropolis.
4. Répéter 2 et 3 Jusqu'à ce que l'équilibre statistique soit atteint.
5. Décroitre la température et répéter Jusqu'à ce que le système soit gelé [40].

Dans un premier temps, T étant généralement choisi très grand, beaucoup de solutions même celles dégradant la valeur de F sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des solutions. Mais à mesure que la température baisse, la plupart des solutions augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique.

A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objectif. Il laisse ainsi une chance au système de

s'extraire d'un minima local. Notons aussi que si la température est égale à 0, seules les solutions optimisant f sont acceptées. L'algorithme se comportera donc comme la méthode de la descente du gradient [39].

Données : L , T, T_0, λ, i % T est la température initial du système, T_0 est la température critique, λ est la raison de la suite de la température, L contient les villes à parcourir avce des informations sur les distances entre les villes
Résultat : L′ : liste % contient la liste du parcours de villes à effectuer
début
  L′ ← **Générer-trajet-initial** S
  T ← L′
  pour k = 1 *à* i **faire**
    tant que T > T_0 **faire**
      L″ ← **transformation** L′
      si *Trajet* L″ < *Trajet* L′ **alors**
        L′ ← L″
      fin
      sinon
        Tirage d'un nombre p ∈ [0; 1]
        si $p > e^{-\frac{Trajet\ L'' - Trajet\ L'}{T}}$ **alors**
          L′ ← L″
        fin
      fin
    fin
    si L′ < T **alors**
      T ← L′
    fin
  fin
  Retourner T
fin

Algorithme 4: Algorithme de Recuit Simulé

FIGURE 1.16 – L'algorithme du recuit simulé[10]

1.9.4.1 Etat initial de l'algorithme

La solution initiale peut être prise au hasard dans l'espace des solutions possibles, elle peut aussi être générée par une heuristique classique, telle que la descente du gradient ou l'algorithme glouton (dans le cas du voyageur du commerce).

La température initiale doit être assez élevée, car c'est elle qui fixe la probabilité d'accepter ou de refuser les solutions défavorables à l'optimisation de la fonction f [39].

1.9.4.2 Variation de la température

Deux approches sont possibles pour décroître la température :

- Décroissance par paliers : Pour chaque valeur de la température, on itère l'algorithme de Metropolis jusqu'à atteindre un équilibre statistique, puis on diminue la température.
- Décroissance continue : On fait baisser la température de manière continue, et la méthode la plus courante consiste à utiliser la formule suivante : $T_{i+1} = \alpha \cdot \frac{T_i}{\alpha} < 1$ (en général, α est compris entre 0.9 et 0.99).

- En effet, s'il est choisi trop grand, la température diminuera très rapidement, ce qui peut entraîner le blocage de l'algorithme dans un minimum local .
- En revanche, s'il est choisi trop petit, la température diminuera très lentement, ce qui prolongera le temps de calcul. [40]

1.9.5 Fonctionnement de l'algorithme de recuit simulé

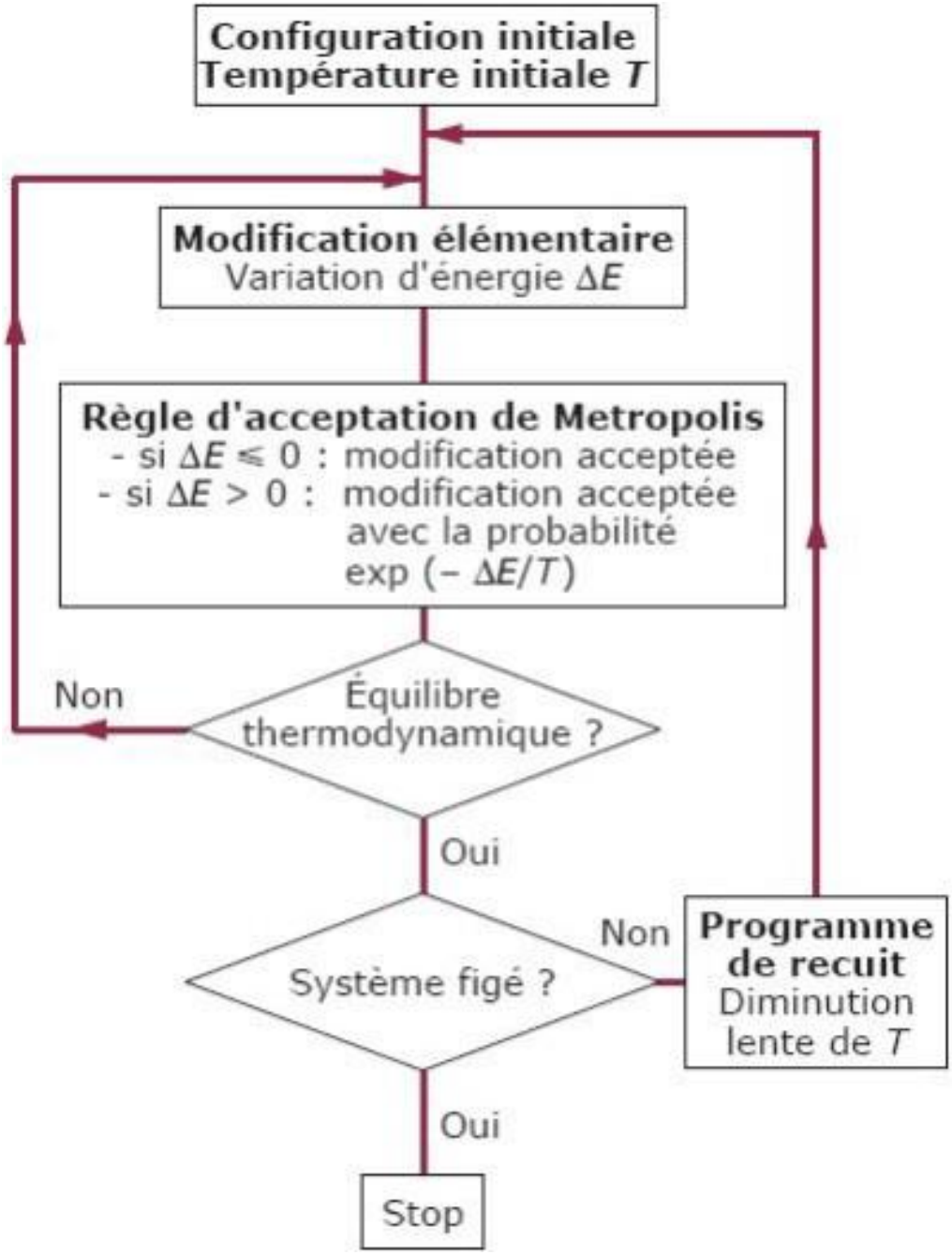


FIGURE 1.17 – L'algorithme du recuit simulé[11]

1.9.6 Application sur un exemple

Le Problème du Voyageur de Commerce (PVC) est un problème d'optimisation combinatoire NP-difficile, dans lequel un voyageur doit visiter un ensemble de villes une seule fois et retourner à la ville de départ en minimisant la distance totale parcourue.

L'implantation du Recuit Simulé pour le PVC consiste à utiliser une méthode d'optimisation stochastique pour explorer l'espace des solutions possibles afin de trouver une solution optimale ou quasi-optimale. Le Recuit Simulé est une méthode d'optimisation stochastique basée sur l'imitation de la physique statistique [39].

Voici les étapes à suivre pour l'implantation du Recuit Simulé pour le PVC :

- **Générer une solution initiale :** Il existe plusieurs façons de générer une solution initiale. Une solution initiale peut être générée de manière aléatoire ou en utilisant une heuristique telle que l'algorithme du plus proche voisin.
- Calculer la fonction objectif : La fonction objectif est la distance totale parcourue par le voyageur. Pour chaque solution, la fonction objectif est calculée en additionnant les distances entre chaque paire de villes dans l'ordre spécifié par la solution.
- La température initiale est un paramètre clé dans le Recuit Simulé. Elle détermine la probabilité d'accepter des solutions moins bonnes que la solution actuelle. Répéter les étapes suivantes jusqu'à ce qu'un critère d'arrêt soit atteint :
    1. Générer une nouvelle solution voisine : Pour générer une nouvelle solution voisine, une ou plusieurs villes sont échangées entre deux positions de la solution actuelle.
    2. Calculer la fonction objectif pour la nouvelle solution : La fonction objectif est recalculée pour la nouvelle solution voisine.
    3. Calculer la variation d'énergie : La variation d'énergie est la différence entre la fonction objectif de la solution actuelle et la fonction objectif de la nouvelle solution voisine.
    4. Décider d'accepter ou de rejeter la nouvelle solution : La probabilité d'accepter une solution moins bonne que la solution actuelle est déterminée par la fonction d'acceptation du Recuit Simulé, qui dépend de la température actuelle et de la variation d'énergie. Si la nouvelle solution est acceptée, elle devient la solution actuelle. Sinon, la solution actuelle reste inchangée.
    5. Mettre à jour la température : La température est mise à jour en fonction d'une fonction de refroidissement.
- Retourner la meilleure solution trouvée : À la fin de l'algorithme, la meilleure solution trouvée est retournée [40].

1.9.7 Domaines d'applications

Comme toute méta-heuristique, la méthode du recuit simulé peut être appliquée à de nombreux problèmes d'optimisation. Les chercheurs l'ont principalement utilisée dans les domaines suivants :

- La conception des circuits intégrés (Kirkpatrick, et al., 1988) pour les problèmes de placement et de routage.
- Le routage des paquets dans les réseaux.
- La segmentation d'images.
- Le problème du voyageur de commerce.
- Le problème du sac à dos[39].

1.9.8 Avantages et Inconvénients

1.9.8.1 Avantages :

- Facile à implémenter.
- Fournit généralement de bonnes solutions par rapport aux algorithmes de recherche classiques.
- Le recuit simulé peut être utilisé dans la plupart des problèmes d'optimisation.
- Il converge vers un optimum global lorsque le nombre d'itérations tend vers l'infini.

Cela en fait une option attrayante pour les problèmes d'optimisation difficiles[39].

1.9.8.2 Inconvénients

La difficulté de déterminer la température initiale :

- Si elle est trop basse, la qualité de la recherche sera mauvaise.
- Si elle est trop élevée, le temps de calcul sera élevé. L'impossibilité de savoir si la solution trouvée est optimale.

Dégradation des performances pour les problèmes comportant peu de minima locaux (comparé aux heuristiques classiques telles que la descente du gradient, par exemple)[40].

1.10 Conclusion

Dans ce chapitre, nous avons présenté les deux principales classes de méthodes de résolution : les méthodes exactes et les méthodes approchées.

Nous avons également souligné que le problème d'affectation des produits aux machines est considéré comme un problème NP-difficile, ce qui signifie qu'il est très difficile de trouver une solution exacte en temps raisonnable pour des instances de grande taille.

C'est pourquoi nous avons présenté plusieurs méthodes de résolution heuristiques, telles que la méthode du recuit simulé. Cette méthode permet de trouver des solutions de qualité en temps raisonnable, même pour des instances de grande taille.

Dans le chapitre suivant, nous décrirons la conception de notre système.

Chapitre 2

Conception

2.1 Introduction

L'affectation d'un produit à une machine est un problème d'optimisation combinatoire qui implique l'assignation d'un produit à une machine tout en maximisant un critère de performance prédéfini. Pour résoudre ce problème, diverses techniques d'optimisation peuvent être utilisées, telles que l'approche du recuit simulé.

L'objectif fondamental consiste à minimiser le temps ou le coût de fabrication du produit en l'attribuant à une machine spécifique. Pour plus d'informations détaillées sur cette problématique, vous pouvez consulter le chapitre 1.

Dans ce chapitre, nous allons présenter notre proposition de conception. Celle-ci se compose d'une modélisation du problème d'affectation des produits aux machines en utilisant la méthode de résolution de problème "Recuit Simulé".

2.2 Conception générale du système

Nous avons tenté de concevoir une solution pour le problème d'affectation de produits aux machines pour le placement de composants en utilisant une architecture à trois couches, qui est représentée dans la figure suivante.

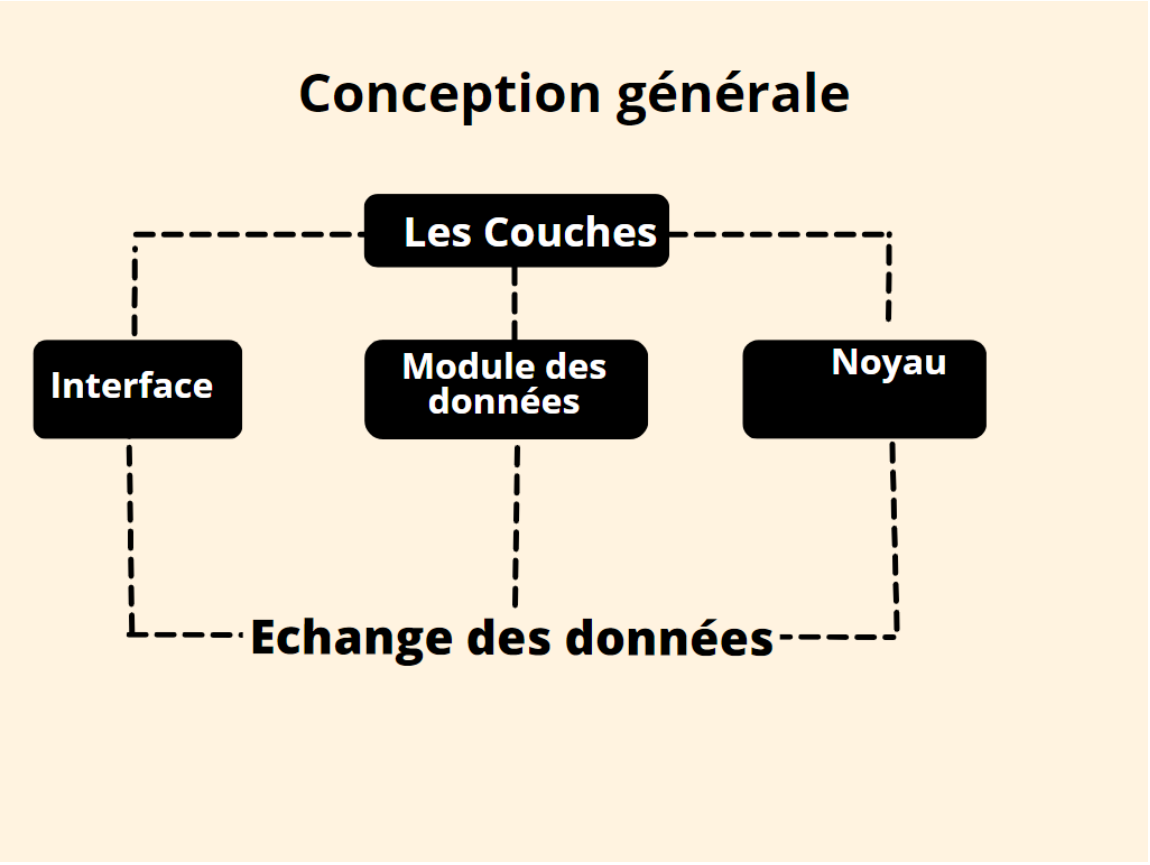


FIGURE 2.1 – Conception générale en trois couches

2.2.1 La couche Interface

La couche Interface ou bien la couche d'interaction avec l'utilisateur c'est la couche qui interagit avec les utilisateurs par le biais d'écrans, de formulaires, de menus, de rapports, etc. C'est la couche la plus visible de l'application. Elle définit l'aspect de l'application [41].

2.2.2 La couche module de données

La couche de données contient des données d'application et une logique métier. La logique métier donne de la valeur à votre application. Elle se compose de règles métier réelles qui déterminent la manière dont les données d'application doivent être créées, stockées et modifiées. Cette séparation permet d'utiliser la couche de données sur plusieurs écrans, de partager des informations entre différentes parties de l'application et de reproduire la logique métier en dehors de l'UI pour les tests unitaires [42].

2.2.3 La couche noyau

La couche du noyau de l'application Cette couche contient les principaux programmes, les définitions du code et les fonctions de base de l'application. Les programmeurs travaillent la plupart du temps sur cette couche [41].

2.3 Conception détaillé de système

2.3.1 La couche interface

L'architecture de la couche interface pour un problème d'affectation de produits aux machines comprend l'entrée de données, saisir les paramètres de la méthode et la présentation des résultats.

Cette architecture permettra aux utilisateurs de résoudre efficacement et facilement le problème d'affectation de produits aux machines en utilisant un solveur d'optimisation.

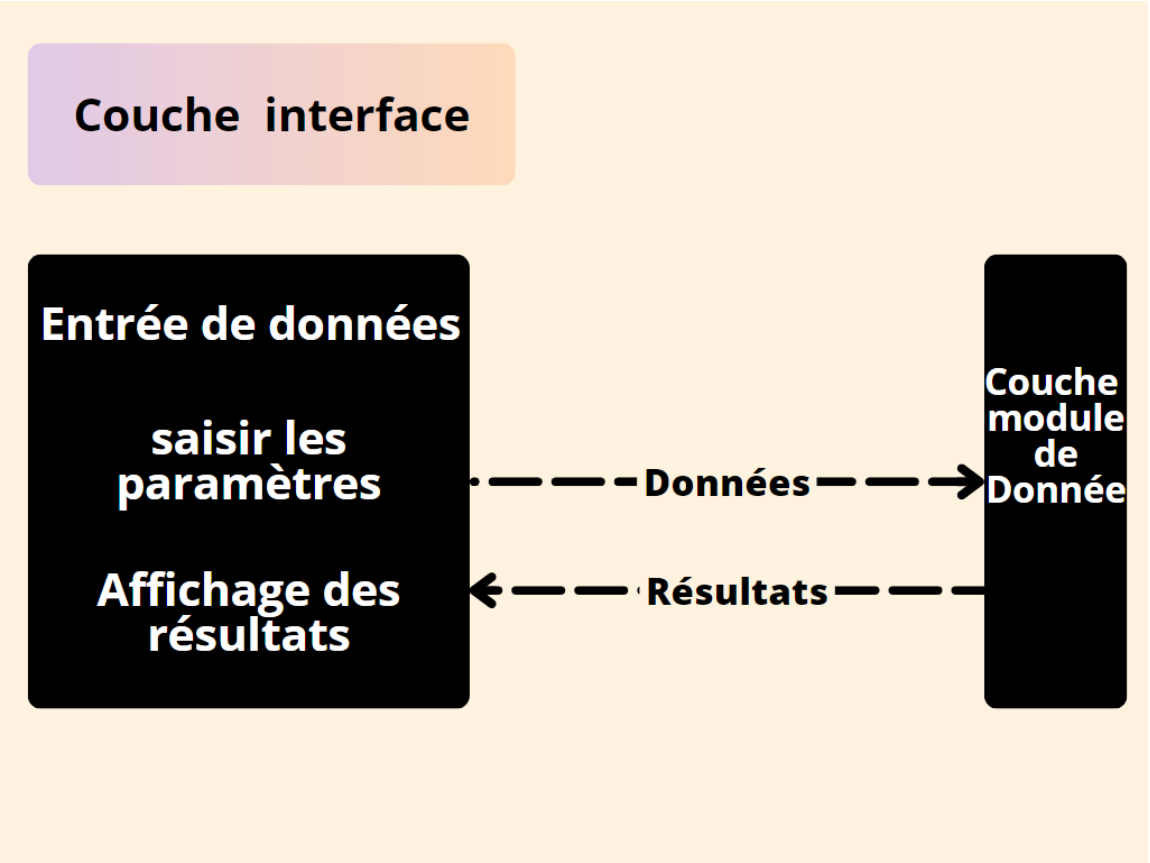


FIGURE 2.2 – L'architecture de la couche interface

2.3.1.1 Entrée de données

La couche interface recevra les données en entrée sous forme des tableaux contenant des informations sur les produits, les machines (les nombres) et le type de génération de l'affectation (aléatoire ou bien à saisir).

2.3.1.2 Saisir les paramètres de la méthode

L'utilisateur doit saisir les paramètres de la méthode de recuit simulé : la température initiale et le nombre d'itérations.

2.3.1.3 Affichage de résultats

Les résultats seront présentés à l'utilisateur sous forme de tableaux et de graphiques afin de permettre une analyse plus approfondie des résultats obtenus.

2.3.1.4 Module graphique

Nous envisageons de développer un module graphique permettant d'afficher plusieurs graphes liés au problème d'affectation de produits aux machines. Voici une description des trois graphes que nous prévoyons d'inclure :

- **Graphe de l'affectation générale :** Ce graphe affiche toutes les possibilités d'affectation des produits aux machines. Chaque nœud du graphe représente une combinaison d'affectation de produits aux machines.

- **Graphe de la solution initiale :** Ce graphe affiche une affectation aléatoire des produits à une seule machine. Chaque nœud représente un produit, et les arêtes représentent les coûts d'affectation à la machine sélectionnée. Ce graphe permettra de visualiser la solution initiale avant l'application de l'algorithme de recuit simulé.
- **Graphe de la meilleure solution :**Ce graphe affiche l'affectation optimisée des produits aux machines après l'exécution de l'algorithme de recuit simulé. Chaque nœud représente un produit et la machine à laquelle il est affecté, et les arêtes représentent les coûts associés. Ce graphe permettra de visualiser la meilleure solution obtenue et les améliorations par rapport à la solution initiale.

Ces graphes seront affichés dans le module graphique, offrant une représentation visuelle des affectations et des coûts associés. Cela permettra aux utilisateurs de mieux comprendre les résultats de l'algorithme de recuit simulé et d'évaluer les performances de différentes solutions.

2.3.2 La couche module de données

L'architecture de la couche module de données pour un problème d'affectation de produits aux machines comprend généralement.

- Un module de structuration de données qui prépare les données pour l'affectation.
- Un module d'affectation de produits aux machines qui utilise une méthode appropriée pour réaliser l'affectation.
- Un module d'échange de données qui permet la communication entre les différents modules.

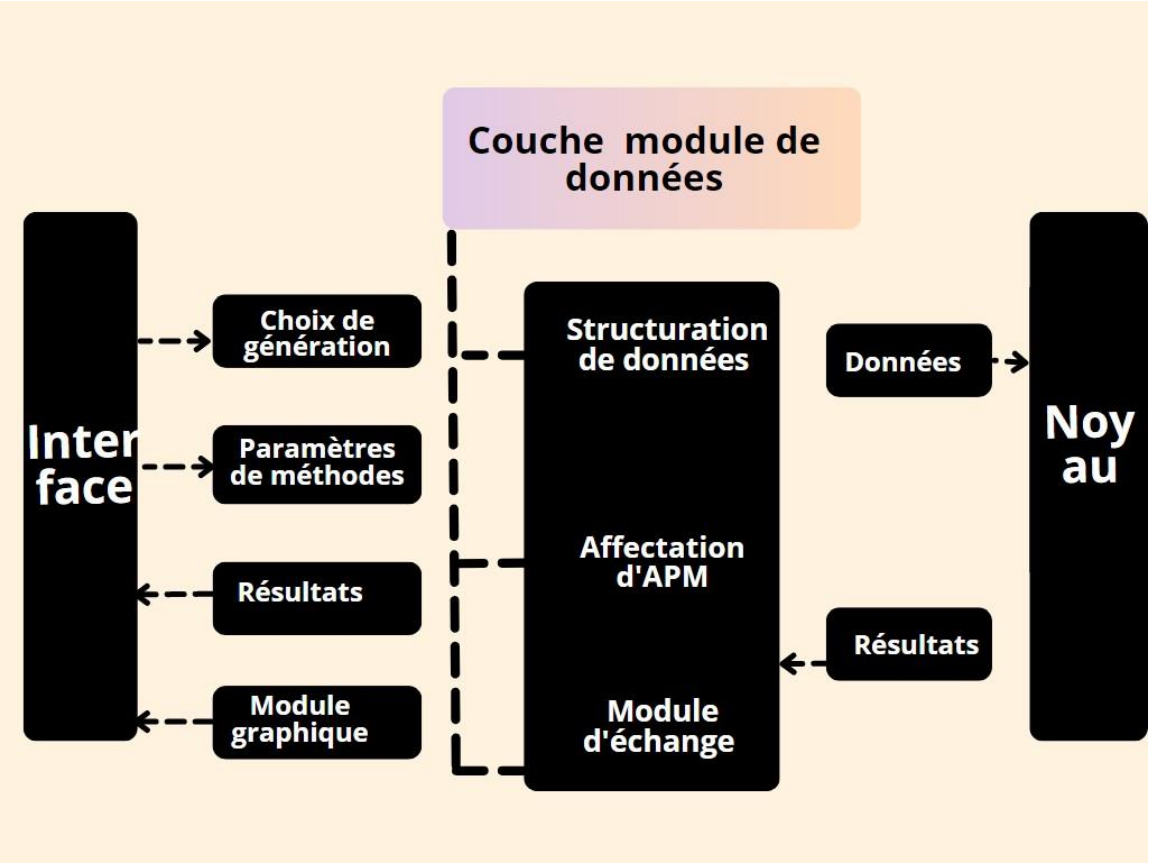


FIGURE 2.3 – L'architecture de la couche module de données

2.3.2.1 Module de structuration de données

C'est un responsable de préparer les données en vue de leur utilisation dans la résolution du problème d'affectation. Ce module peut effectuer des tâches telles que la normalisation des données , la transformation des données, la détection et le nettoyage des données manquantes ou aberrantes, etc.

2.3.2.2 Module d'affectation de produits aux machines

Ce module est responsable de l'affectation des produits aux machines en utilisant une méthode appropriée. Il peut utiliser différentes méthodes d'affectation.

Le module d'affectation doit prendre en compte les caractéristiques des produits et des machines, ainsi que les contraintes à respecter (par exemple, une seule capacité de machine peut être affectée à un seul produit), afin d'optimiser l'affectation des produits aux machines.

2.3.2.3 Le module d'échange de données

Il est également important dans cette architecture, car il permet aux différents modules de communiquer entre eux. Ce module est responsable de la transmission des données collectées et prétraitées et des résultats.

2.3.3 La couche noyau

L'importance de cette couche est primordiale dans notre outil, elle représente l'implémentation des méthodes de résolution. Son rôle est d'exploiter les structures de données et paramètres en provenance de la couche échange pour résoudre le problème avec une méthode de résolution et de renvoyer les résultats obtenus[43].

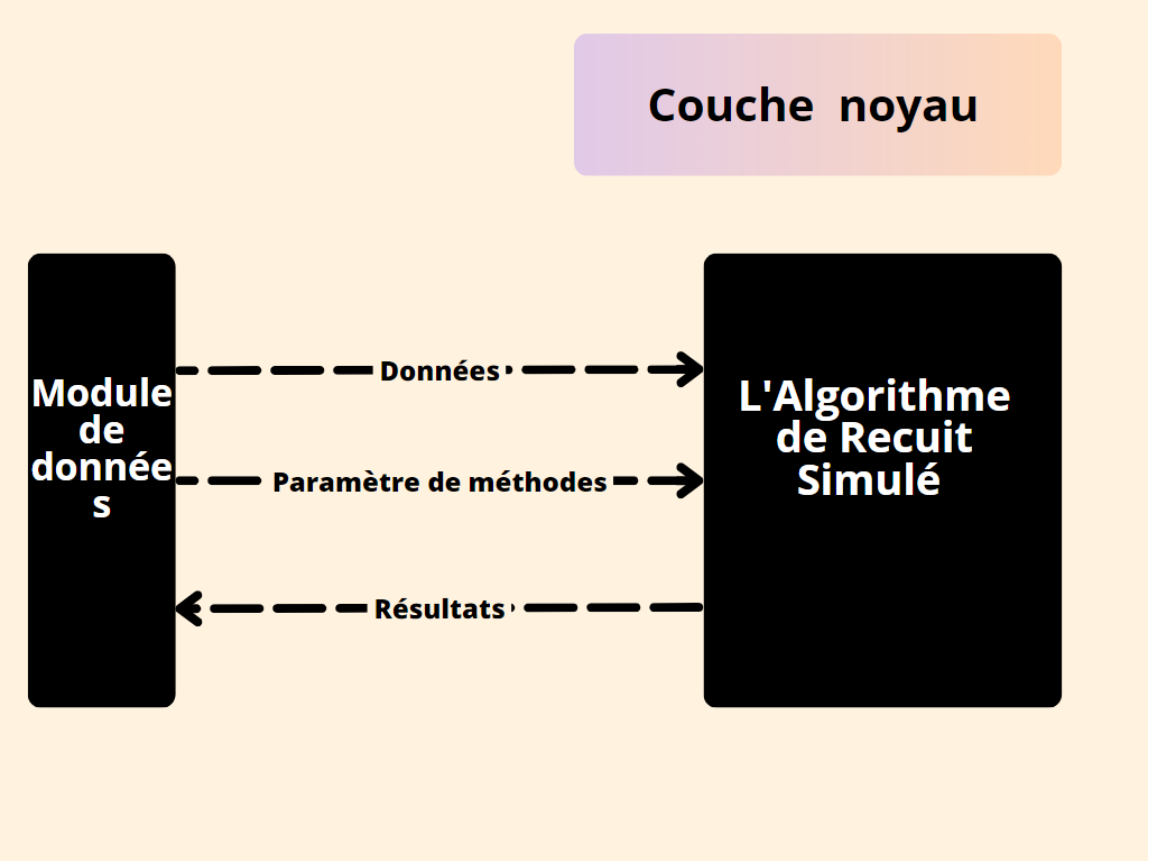


FIGURE 2.4 – L'architecture de la couche noyau

2.4 L'architecture d'algorithme recuit simulé

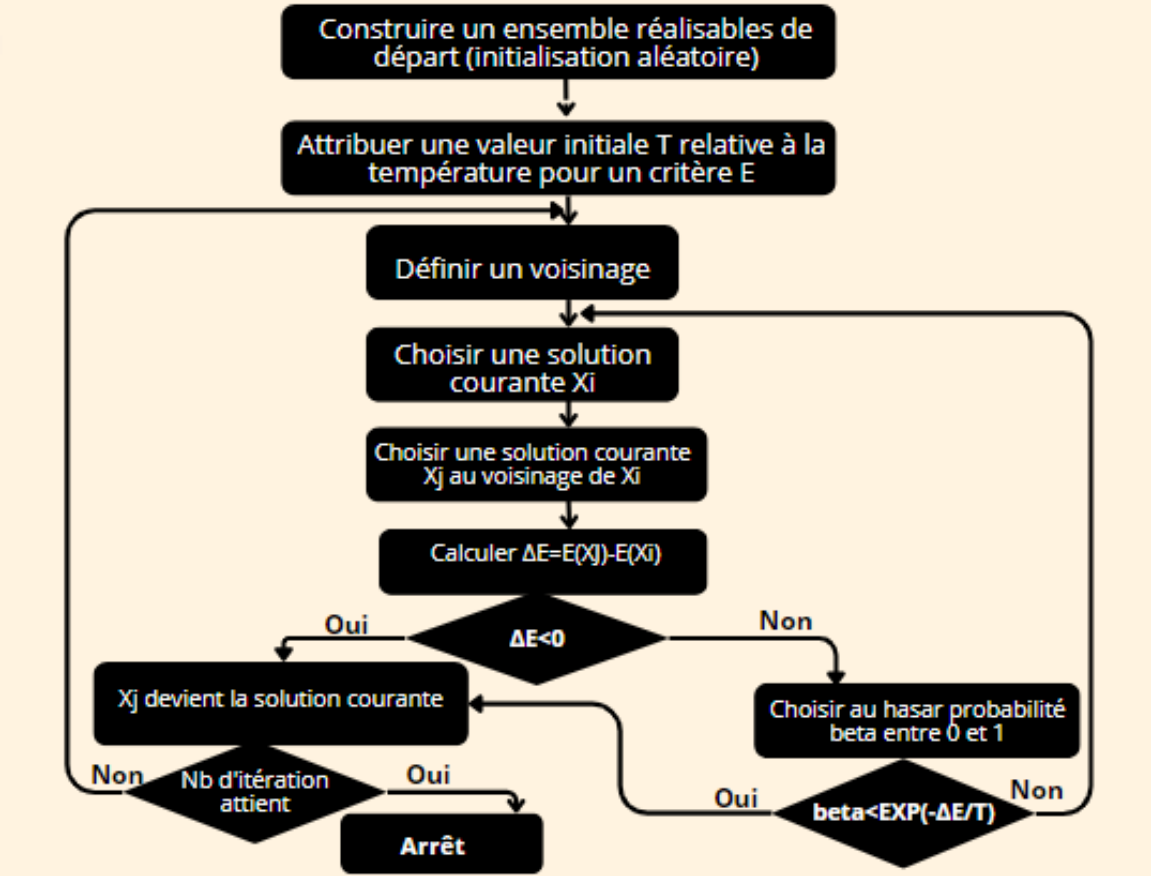


FIGURE 2.5 – Architecture d'algorithme recuit simulé[12]

2.5 L'adaptation de RS au problème d'APM

L'adaptation de l'algorithme de Recuit simulé au problème d'affectation de produits aux machines est une méthode efficace pour résoudre ce type de problème d'optimisation combinatoire.

Elle permet de trouver des solutions de bonne qualité en explorant l'espace de recherche de manière probabiliste et en utilisant une température qui contrôle la probabilité de transition entre les configurations.

2.5.1 Les paramètres de L'algorithme du recuit simulé

- La température T = liste de réels positifs.
- La température permet de contrôler l'acceptation des solutions ou non par le calcul du critère de Métropolis exp $= \frac{\Delta f}{T}$.
- La valeur du paramètre température varie au cours de la recherche des itérations.
- Elle diminue au fur à mesure pour atteindre la valeur 0.
- Beta : test de la probabilité d'accepter la solution ou non.

- X0 : solution initiale.
- T0 : température initiale (très grande).
- Choisie un test d'arrêt . [8]

2.5.2 Codage des solutions

Le recuit simulé est un algorithme itératif qui utilise le critère de Metropolis exp $\frac{-\Delta E}{T}$. Il est composé de deux boucles imbriquées :

- La première boucle sert à abaisser la température, ce qui favorise la stabilité thermodynamique. L'objectif est de converger vers l'optimum global (le minimum global dans le cas du problème d'affectation de produits aux machines).
- La deuxième boucle est utilisée pour sélectionner le meilleur voisin de la solution courante en se basant sur le critère de Metropolis.

2.5.3 La génération des affectations produit-machine

L'algorithme de recuit simulé peut être utilisé pour résoudre le problème d'affectation de produits aux machines en tenant compte de l'affectation générale des coûts. L'affectation générale des coûts peut être réalisée de deux manières : de manière aléatoire ou en permettant à l'utilisateur de saisir les valeurs.

Produit / Machine	Produit1	Produit2	Produit3
Machine1	coût1_1	coût1_2	coût1_3
Machine2	coût2_1	coût2_2	coût2_3
Machine3	coût3_1	coût3_2	coût3_3

FIGURE 2.6 – La génération des affectations produit-machine

2.5.3.1 Affectation aléatoire

Dans le cas d'une affectation aléatoire des coûts, les coûts d'affectation des produits aux machines sont générés de manière aléatoire, tout en respectant les contraintes et les limites du problème. Cette approche permet d'explorer différents scénarios et de tester la robustesse de l'algorithme face à des variations aléatoires des coûts. Chaque produit peut être affecté à une machine unique.

2.5.3.2 Affectation par saisie utilisateur

Dans le cas où les coûts d'affectation sont saisis par l'utilisateur, ce dernier peut fournir les valeurs de coûts spécifiques pour chaque affectation produit-machine. Cela permet de prendre en compte des informations spécifiques et des priorités particulières dans le processus d'affectation.

Il est important de noter que l'affectation générale des coûts joue un rôle clé dans la performance de l'algorithme de recuit simulé. Des coûts d'affectation bien ajustés peuvent conduire à des solutions de meilleure qualité et à une convergence plus rapide vers l'optimum global.

2.5.4 Définition d'une solution initiale

La solution initiale de l'algorithme de recuit simulé pour un problème d'affectation de produits aux machines peut être représentée par une matrice N x M, où N représente le nombre de produits et M représente le nombre de machines.

Chaque élément de la matrice représente l'affectation d'un produit à une machine (Chaque produit est affecté à une seule machine). Si le produit i est affecté à la machine j, l'élément (i, j) de la matrice contient la valeur 1 ; sinon, il contient la valeur 0.

Produit / Machine	Produit1	Produit2	Produit3
Machine1	0	1	0
Machine2	1	0	0
Machine3	0	0	1

FIGURE 2.7 – Codage d'une solution initiale

2.5.5 Le choix de la température

2.5.5.1 Température initiale

La température initiale doit être choisie suffisamment élevée pour que l'équilibre thermique soit rapidement atteint. Cela signifie qu'elle doit être grande par rapport aux variations aléatoires d'énergie du système lors de la recherche de l'équilibre.

2.5.5.2 La décroissance de la température

La décroissance de la température dans l'algorithme du recuit simulé est une étape cruciale pour trouver un compromis entre la recherche du meilleur résultat possible et le temps de calcul. Il est courant d'utiliser une loi de décroissance exponentielle pour la température, telle que :

$T_{k+1} = T_k \cdot \exp(-\tau)$, où τ est un paramètre choisi de manière à garantir une décroissance suffisamment lente de la température.

Si nous prévoyons d'implémenter l'algorithme du recuit simulé en utilisant Java Swing, nous pouvons intégrer la fonction `Math.random()` dans notre code. La fonction `Math.random()` est une fonction intégrée de Java qui génère des nombres aléatoires compris entre 0 et 1.

2.5.5.3 Température finale

C'est un compromis entre la qualité du résultat et le temps de calcul, et cela dépend de la nature du problème d'optimisation. On fait varier les valeurs finales (minimales) [1].

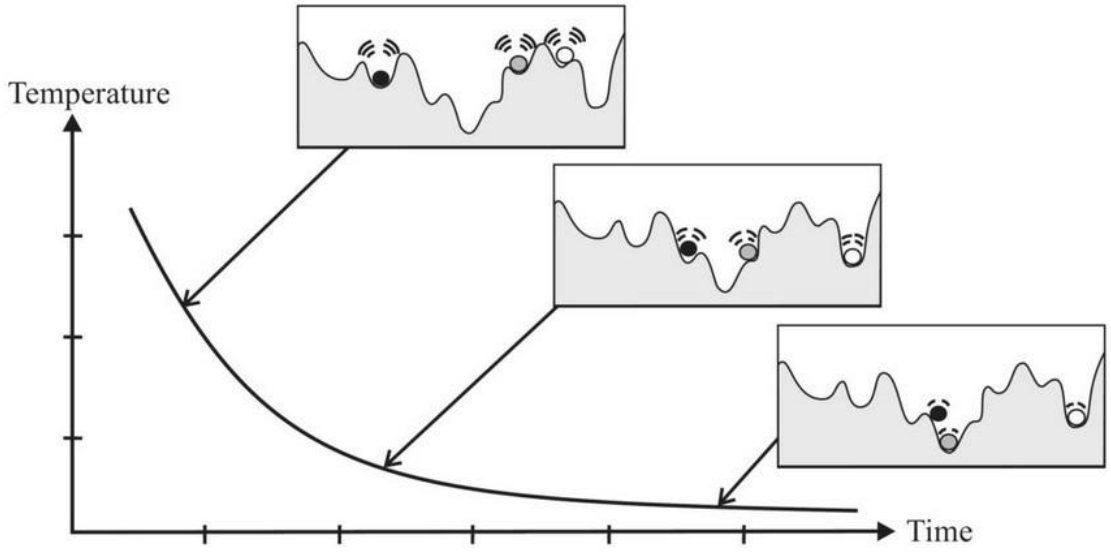


FIGURE 2.8 – La décroissance de la température [13]

2.5.6 La fonction de coût(énergie)

- La fonction objectif (ou énergie) E à minimiser peut être définie comme la somme des coûts d'affectation pour toutes les machines M et tous les produits P.
- L'énergie (E) est définie comme la minimisation de la somme des coûts d'affectation : $E = \sum_i \sum_{jm} \sum_{jp} C_{ij}$.
- Les états du système peuvent être définis comme les différentes affectations possibles des produits aux machines.

- L'énergie peut être calculée en utilisant le temps de traitement total pour chaque machine. Cette énergie représente la somme des coûts d'affectation pour chaque affectation produit-machine.
- Il est important de noter que les coûts d'affectation (C_{ij}) peuvent varier en fonction de différents facteurs, tels que la capacité de la machine à affecter un seul produit à la fois.
- L'objectif est de trouver une affectation des produits aux machines qui minimise les coûts d'affectation totaux.
- L'algorithme du recuit simulé sera utilisé pour explorer l'espace des solutions en modifiant les affectations des produits aux machines, tout en maintenant la contrainte de minimisation des coûts d'affectation.

la figure suivante présentant le calcul des coûts.

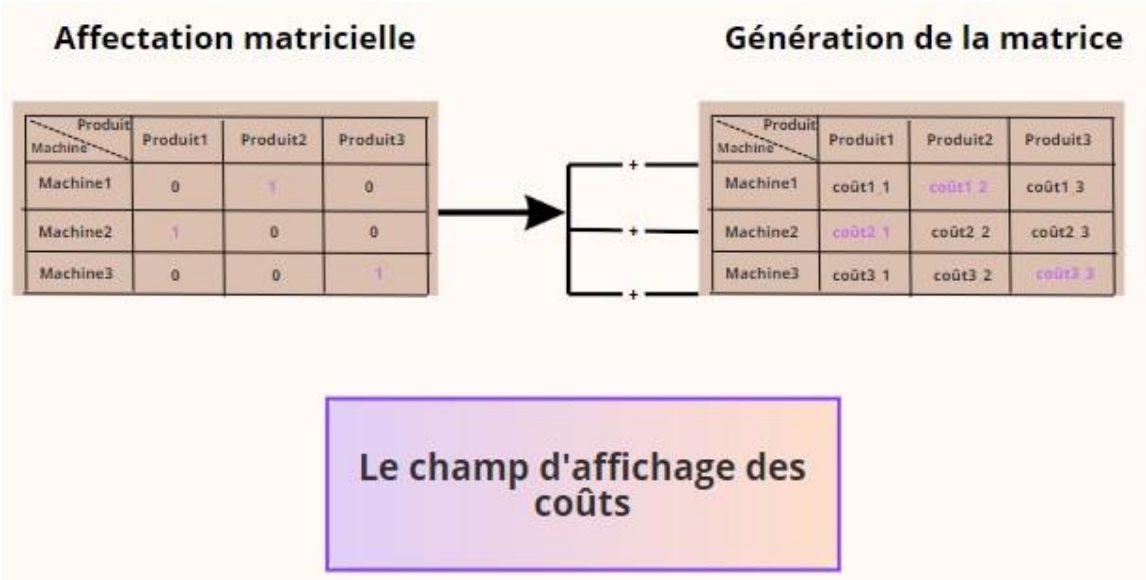


FIGURE 2.9 – Mécanisme de calcul des coûts

2.5.7 Génération de voisins

La génération de voisins est une étape clé de l'algorithme de recuit simulé qui permet d'explorer l'espace des solutions possibles. Pour le problème d'affectation produit aux machines, un voisin peut être obtenu en permutant l'affectation d'un produit entre deux machines.

Il est important de noter que la génération de voisins doit être effectuée de manière à ce que chaque voisin soit valide, c'est-à-dire que chaque produit soit affecté à une seule machine et que chaque machine ne soit pas surchargée. Si un voisin est invalide, il doit être rejeté et un nouveau voisin doit être généré.

2.5.8 Critère d'acceptation

Le critère d'acceptation peut être basé sur la différence de coût entre la solution actuelle et le voisin proposé.

Le coût de la solution actuelle représente la somme des coûts d'affectation de chaque produit à sa machine respective.

Le coût d'un voisin proposé peut être calculé de la même manière.

Si le coût du voisin proposé est inférieur au coût de la solution actuelle, le voisin est généralement accepté.

Si le coût du voisin proposé est supérieur au coût de la solution actuelle, le voisin peut être accepté avec une certaine probabilité, qui diminue au fil du temps.

2.5.9 Le nombre de sauts

Le nombre de sauts aléatoires dépendra de plusieurs facteurs, tels que la taille du problème, la complexité de la fonction de coût, la précision de la solution souhaitée et le temps de calcul disponible.

En général, il est conseillé de commencer avec un nombre relativement élevé de sauts aléatoires pour permettre une exploration plus approfondie de l'espace des solutions. Ensuite, le nombre de sauts peut être progressivement réduit à mesure que la température diminue et que l'algorithme se rapproche de la solution optimale.

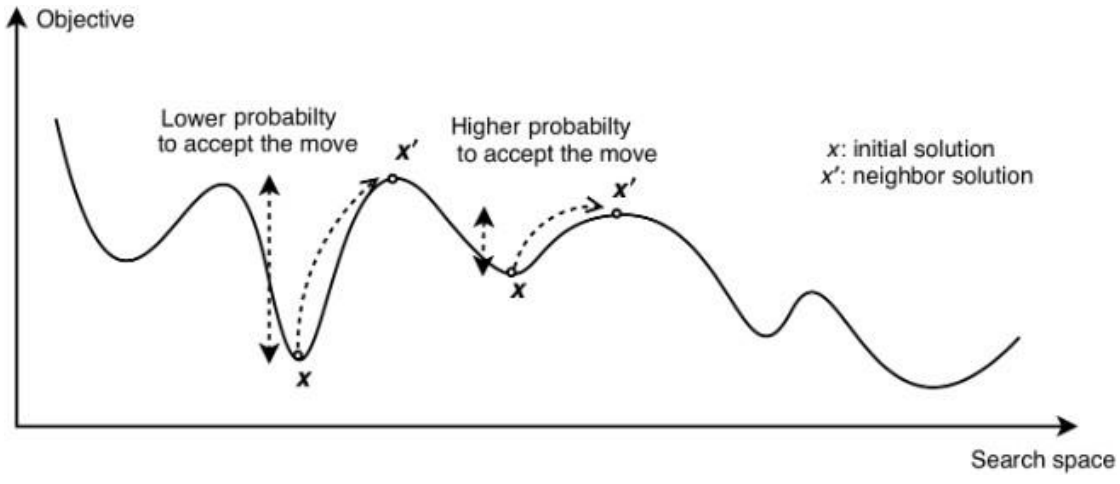


FIGURE 2.10 – Un saut aléatoire pour explorer l'espace des solutions [14]

2.5.10 Le critère d'échange(Permutation d'affectation)

Cette opération consiste à permuter l'affectation d'un produit entre deux machines. Par exemple, si le produit i est affecté à la machine j et le produit k est affecté à la

machine l, la permutation d'affectation consiste à affecter le produit i à la machine l et le produit k à la machine j.

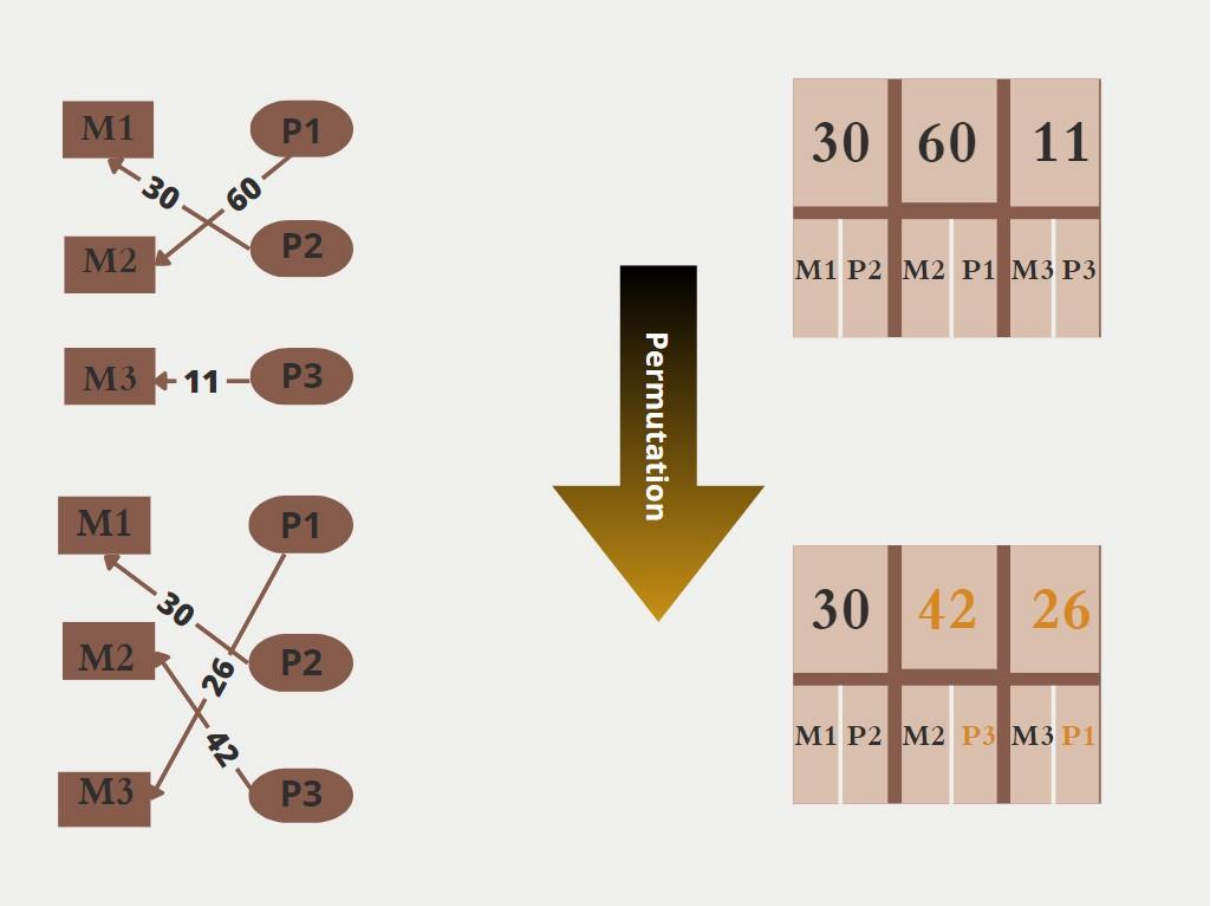


FIGURE 2.11 – Exemple de Permutation

2.5.11 Le critère d'arrêt

Le critère d'arrêt dépend du temps (le nombre d'itérations) et de la dégradation de la solution (Δf), qui est évaluée comme la différence de la fonction coût entre la solution précédente et la nouvelle solution sélectionnée (chaque coût représente une pondération de l'affectation du produit y à la machine x).

À chaque dégradation, l'algorithme s'arrête avec une probabilité dépendant de cette dégradation et du temps (nombre d'itérations) de l'algorithme :

- Plus la dégradation est grande, plus la probabilité de continuer diminue.
- Plus le nombre d'itérations est grand, plus la probabilité de continuer diminue également.

2.6 Conclusion

Le chapitre actuel a exposé une architecture de système qui pourrait mener à une conception adéquate pour résoudre le problème d'affectation de produits aux ma-

chines en appliquant l'algorithme de recuit simulé. Dans le prochain chapitre, les détails de la mise en œuvre de cette conception ainsi que les outils utilisés pour la réalisation du système seront présentés.

Chapitre 3

Implementation

3.1 Introduction

Ce chapitre vise à décrire les différentes étapes de mise en œuvre de notre système qui a été proposé dans le chapitre de conception. Nous commencerons par discuter de l'environnement de développement utilisé. Ensuite, nous présenterons les procédures et la structure des données utilisées dans le système.

3.2 Environnement de développement

Notre système est développé sous l'environnement suivant :

- Micro-portable Dell (i5-3340M CPU 2.70GHZ , RAM 4,00 Go).
- Système d'exploitation Microsoft Windows 10.

Pour développer notre application on a choisi l'environnement Eclipse. Il existe de nombreux avantages à utiliser Eclipse pour le développement de logiciels, en voici quelques-uns :

- Plateforme extensible : Eclipse fournit une plateforme extensible qui permet aux développeurs d'ajouter des fonctionnalités supplémentaires à l'IDE en installant des plugins. Cela permet aux développeurs de personnaliser leur environnement de développement en fonction de leurs besoins [44].

FIGURE 3.1 – Eclipse IDE[15]

3.3 Langage de programmation

Java est un langage de programmation orienté objet populaire et largement utilisé pour le développement d'applications, de sites web et de services. Il est apprécié pour sa simplicité, sa sécurité, sa portabilité et sa capacité à fonctionner sur différentes plates-formes.

Java est également utilisé pour le développement d'applications mobiles Android, de jeux, d'outils de développement et de nombreuses autres applications.

Java a été créé en 1995 par Sun Microsystems et est maintenant développé et soutenu par Oracle corporation [45] .

FIGURE 3.2 – Java-Development using Eclipse IDE[16]

3.4 Présentation de l'application

3.4.1 Interface

Dans notre application, nous avons une interface principale.

3.4.1.1 L'interface principale

En utilisant Java Swing, il est possible de créer une interface utilisateur avec différents éléments pour interagir avec l'application. Nous avons déjà créé une telle interface utilisateur, comme illustré dans la figure suivante.

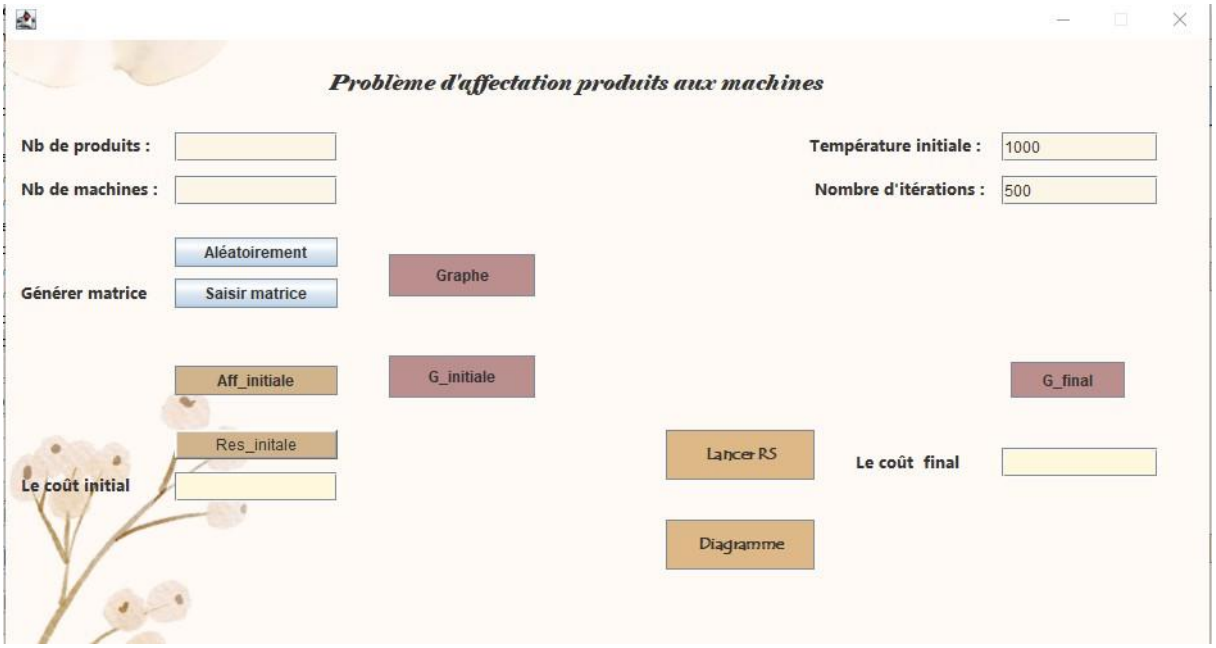


FIGURE 3.3 – L'interface principale

Notre interface principale se compose de plusieurs composants, et nous allons maintenant présenter les rôles de chacun de ces composants.

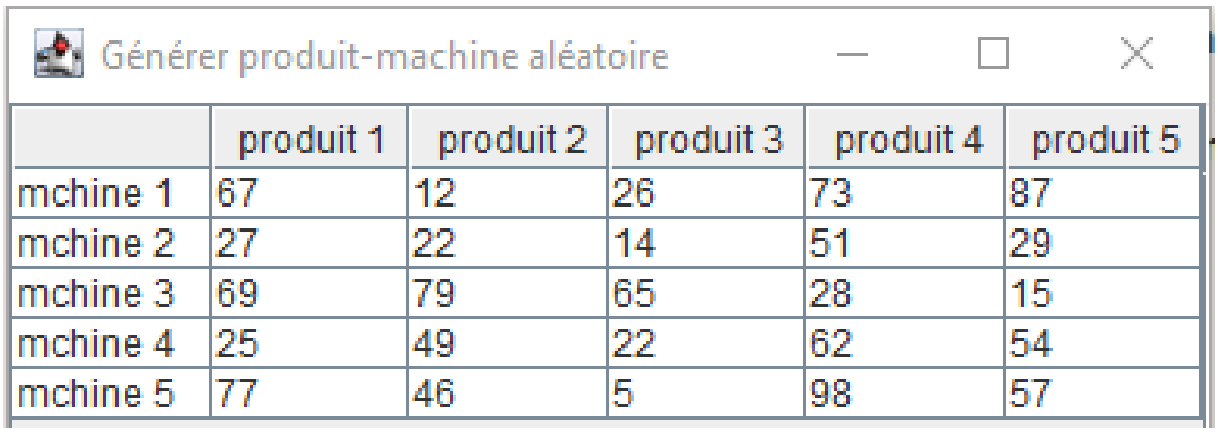
3.4.1.2 Le composant de type de génération

Ce composant se compose de deux champs et trois boutons, qui sont les suivants :

- **Le champ "Nombre de produits" :** Ce champ permet de spécifier le nombre de produits pour l'utilisateur dans la génération aléatoire des affectations des produits aux machines.
- **Le champ "Nombre de machines" :** Ce champ permet de spécifier le nombre de machines à utiliser dans la génération aléatoire des affectations des produits aux machines.

La génération de matrices (aléatoires ou saisies) : Ce bouton permet de choisir le type de génération de la matrice du problème d'APM, que ce soit de manière aléatoire ou en la saisissant manuellement.

- **Le bouton "Générer aléatoirement" :** Ce bouton permet de générer de manière aléatoire les affectations des produits aux machines en déterminant le nombre de machines et de produits.



	produit 1	produit 2	produit 3	produit 4	produit 5
mchine 1	67	12	26	73	87
mchine 2	27	22	14	51	29
mchine 3	69	79	65	28	15
mchine 4	25	49	22	62	54
mchine 5	77	46	5	98	57

FIGURE 3.4 – Générer produit-machine aléatoire

- **Le bouton "Saisir matrice" :** Ce bouton permet à l'utilisateur de générer les affectations des produits aux machines en les saisissant manuellement, en spécifiant le nombre de machines et de produits.

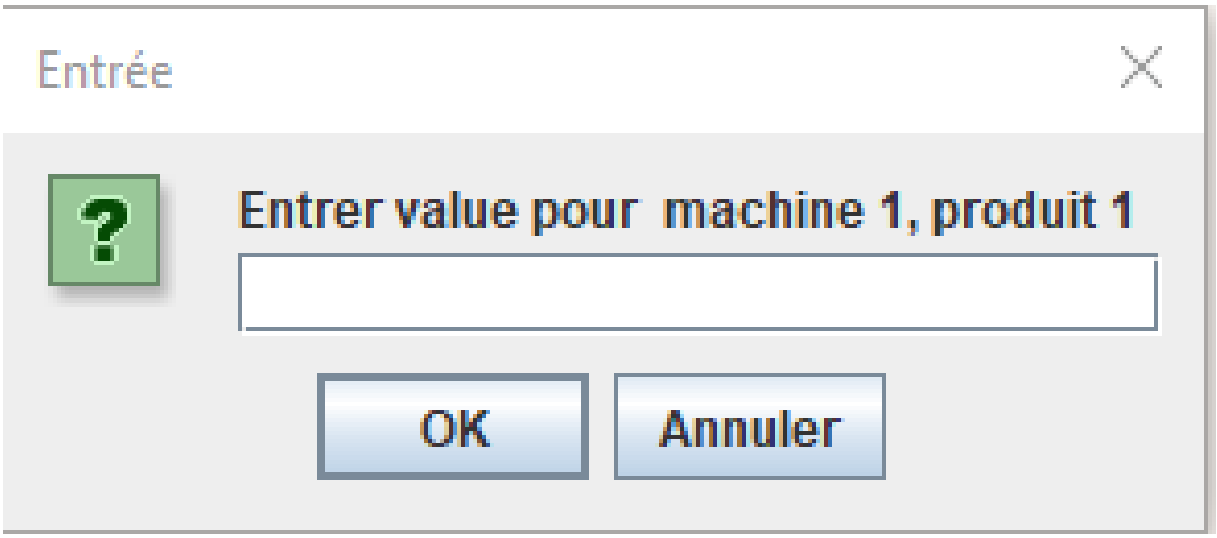


FIGURE 3.5 – Le composant saisie de matrice

- **Le bouton " Graphe" :** Ce bouton permet d'afficher graphiquement le graphe global du problème d'APM qui est généré soit par le mode de génération aléatoire, soit par le mode de génération par saisie.

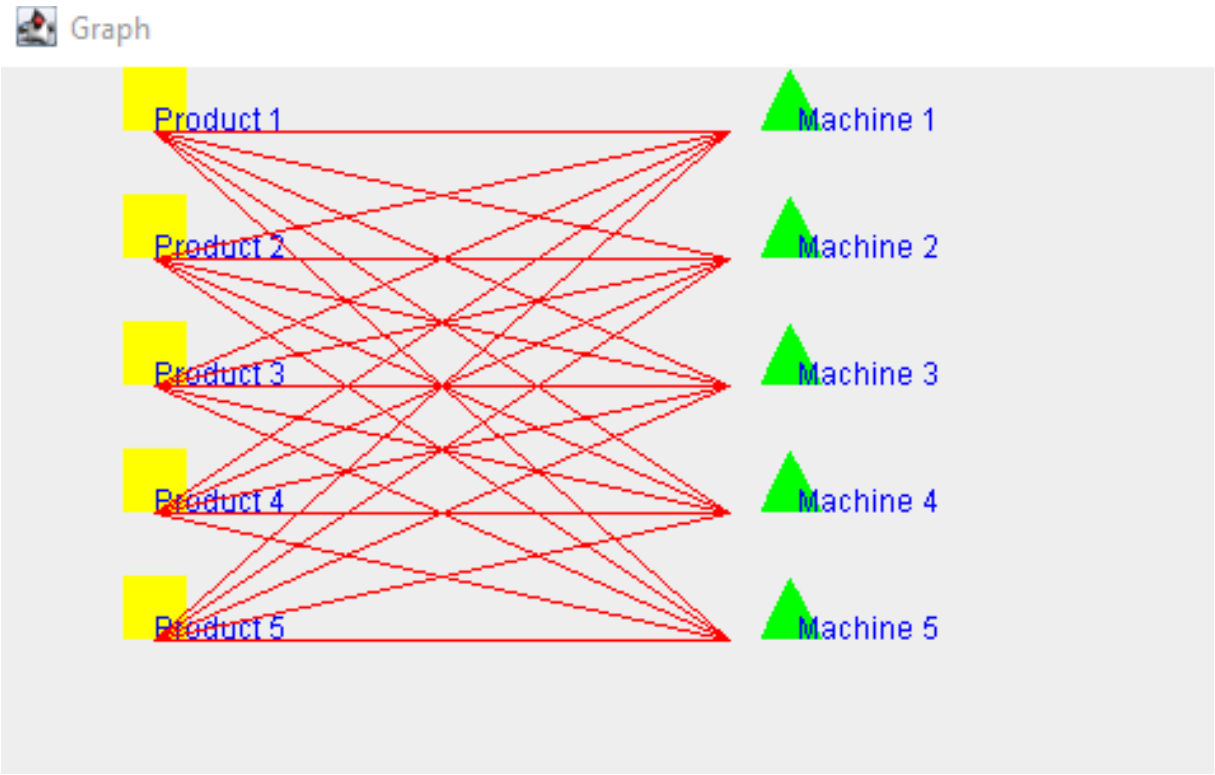


FIGURE 3.6 – Le composant de graphe global

3.4.1.3 Le composant de type d'initialisation

Ce composant se compose d'un champ et de trois boutons, qui sont les suivants :

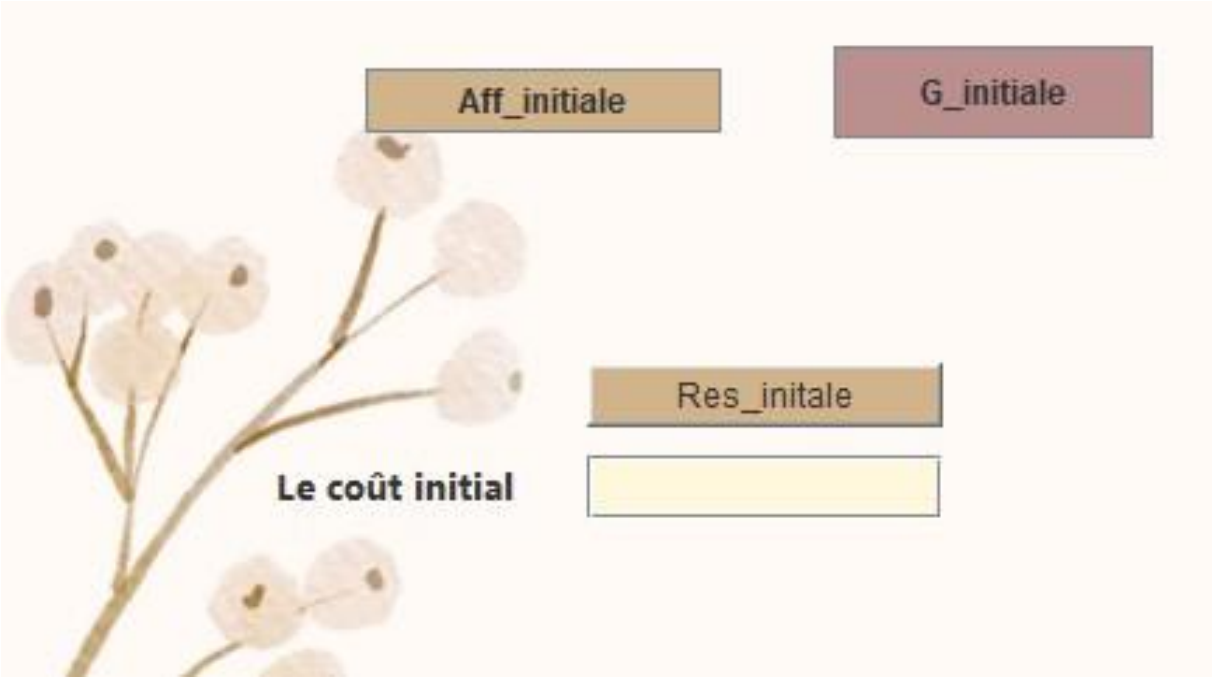
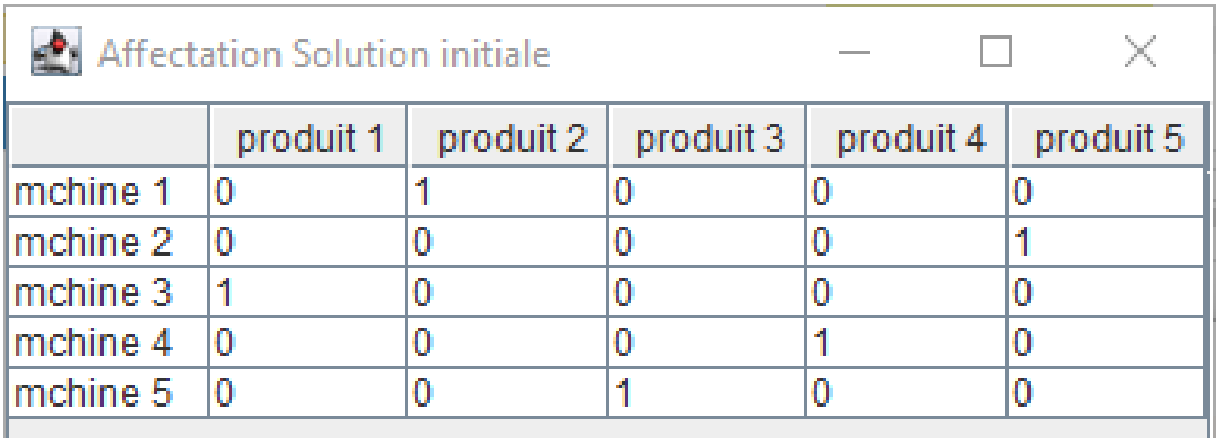


FIGURE 3.7 – Le composant d'initialisation.

- **Le bouton "Aff_ initiale" :** Ce bouton permet de générer une solution initiale aléatoire sous forme d'une matrice binaire de dimensions spécifiées par l'utilisateur, en indiquant le nombre de machines et de produits.



	produit 1	produit 2	produit 3	produit 4	produit 5
mchine 1	0	1	0	0	0
mchine 2	0	0	0	0	1
mchine 3	1	0	0	0	0
mchine 4	0	0	0	1	0
mchine 5	0	0	1	0	0

FIGURE 3.8 – Affectation de solution initiale

- **Le bouton "G_ initial" :** Ce bouton permet de générer un graphe pour la solution initiale en utilisant l'affectation aléatoire et le coût (le temps d'exécution) pour associer la matrice de solution initiale avec la matrice générale.

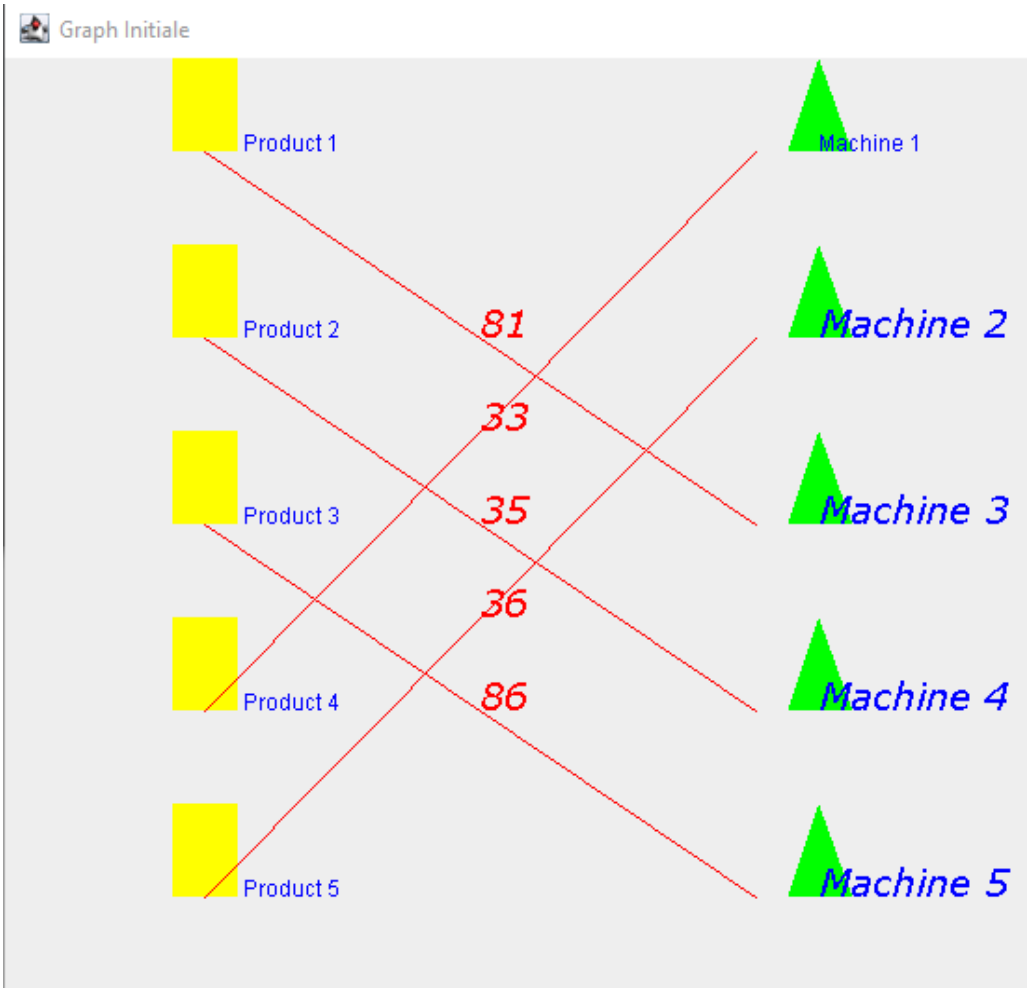


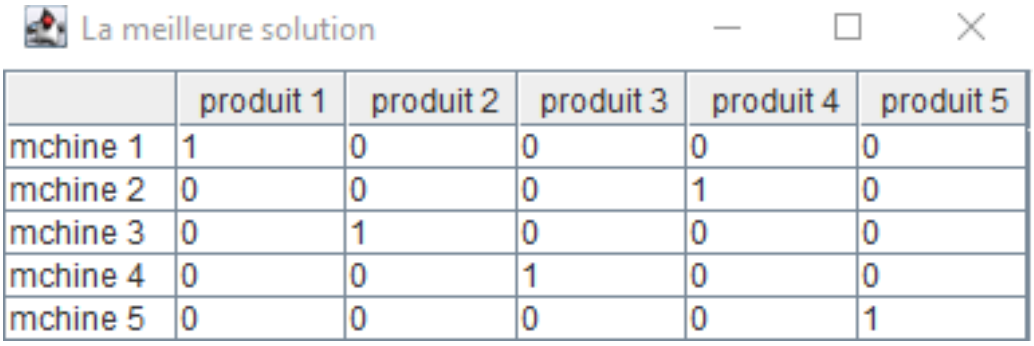
FIGURE 3.9 – La génération du graphe pour la solution initiale

- **Le bouton " Res_ initial" :** Après avoir cliqué dessus, il calcule le coût initial en effectuant la somme des coûts de la solution initiale.
- **Le champ "Le coût initial" :** Ce champ affiche le résultat du coût initial.

3.4.1.4 Le composant de Paramètres de l'algorithme de recuit simulé

se compose de trois champs et deux boutons.

- **Le champ "Température initiale" :** Ce champ permet de spécifier le nombre de température initiale qui va être généré. Ce nombre est déterminé par l'utilisateur.
- **Le champ "Nombre d'itérations" :** Ce champ permet de spécifier le nombre d'itérations. Ce nombre est déterminé par l'utilisateur.
- **Le bouton "Lancer RS" :** Ce bouton permet d'exécuter l'algorithme du recuit simulé, d'afficher la matrice de la meilleure solution obtenue, et ensuite de calculer le coût final après l'optimisation.



	produit 1	produit 2	produit 3	produit 4	produit 5
mchine 1	1	0	0	0	0
mchine 2	0	0	0	1	0
mchine 3	0	1	0	0	0
mchine 4	0	0	1	0	0
mchine 5	0	0	0	0	1

FIGURE 3.10 – La meilleure solution après l'exécution de l'algorithme du RS

- **Le champ "Le coût final" :** Ce champ affiche le résultat du coût final.
- **Le bouton "G_ final" :** Ce bouton permet de générer un graphe pour la meilleure solution afin d'associer la matrice de la solution finale avec la matrice générale.

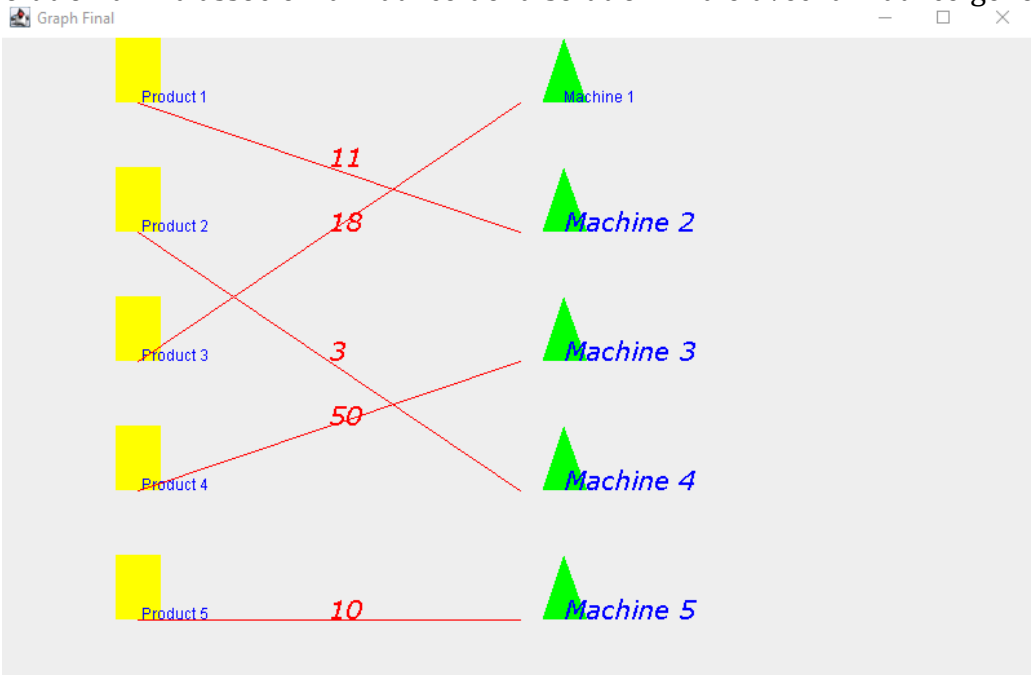


FIGURE 3.11 – La génération du graphe final pour la meilleure solution

3.4.1.5 L'optimisation de l'algorithme de recuit simulé

- **Le bouton"Diagramme"** :En cliquant sur le bouton "Diagramme à bandes", vous pourrez visualiser graphiquement les résultats de cette analyse et optimisation, ce qui facilitera la compréhension des relations entre les paramètres de contrôle et les performances de l'algorithme de recuit simulé.

3.5 Structures de données utilisées

Dans notre application de problème d'affectation des produits aux machines, nous pouvons utiliser différentes structures de données en fonction des besoins et des exigences spécifiques. Voici quelques exemples courants :

3.5.1 Tableau bidimensionnel

Le tableau *matrix* est un tableau à deux dimensions qui est utilisé pour stocker les valeurs de la matrice générée aléatoirement. Chaque élément *matrix*[*i*][*j*] représente une valeur aléatoire de la matrice située à la ligne *i* et à la colonne *j*.

3.5.2 DefaultTableModel

Il s'agit d'une classe fournie par la bibliothèque Swing qui implémente l'interface Table Model et est utilisée pour créer un modèle de tableau par défaut. Dans ce cas, le modèle *model* est utilisé pour créer une JTable avec le nombre de lignes *nbrows* et le nombre de colonnes *nbcolumns*. Les valeurs aléatoires sont ensuite attribuées à chaque cellule du modèle à l'aide de *model.setValueAt(x, i, j)*.

3.5.3 List_u

Une liste des tableaux (*ArrayList*) est utilisée pour stocker les valeurs aléatoires générées pour chaque cellule de la matrice. Chaque valeur *x* est ajoutée à la liste avec *list_u*.

3.5.4 Graphes

Utilisés pour représenter les relations entre les produits et les machines et les coûts.

3.6 Les procédures utilisées

Les procédures utilisées dans notre application de problème d'affectation des produits aux machines sont les suivantes :

3.6.1 Génération aléatoire des affectations :

Cette procédure génère aléatoirement les affectations des produits aux machines en utilisant les nombres de machines et de produits spécifiés par l'utilisateur. Elle utilise des boucles et des conditions pour attribuer les produits aux machines en respectant les capacités des machines.

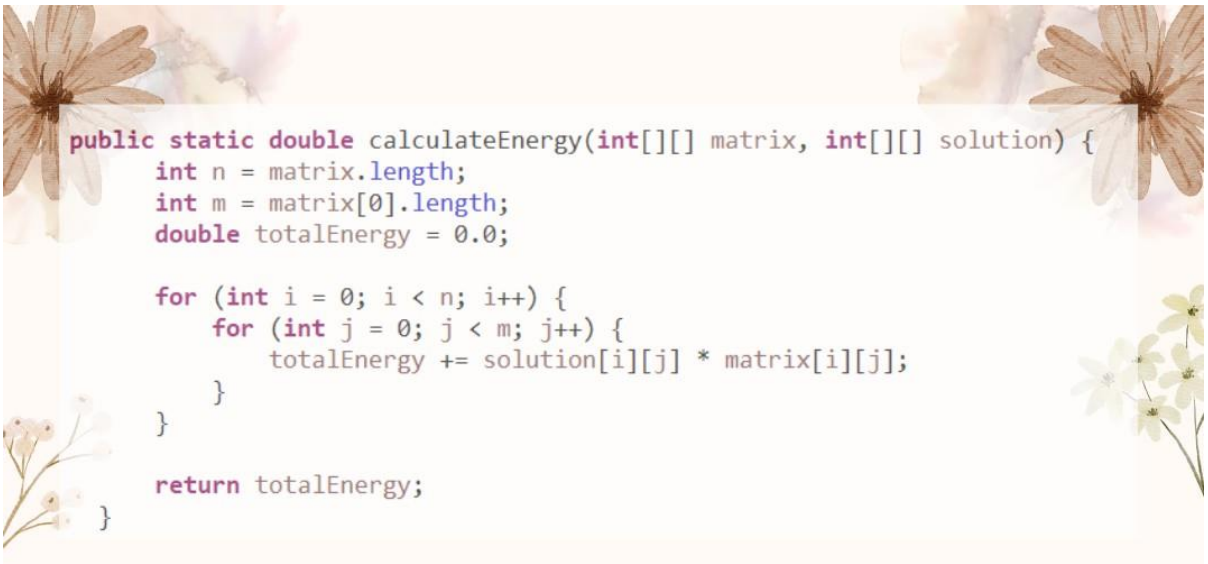
Voici le code :

```
public void actionPerformed(ActionEvent e) {
    int nbrows, nbcolumns;
    nbrows = Integer.valueOf(textField.getText());
    nbcolumns = Integer.valueOf(text.getText());

    DefaultTableModel model = new DefaultTableModel(nbrows, nbcolumns);

    // Remplir la matrice avec des valeurs aléatoires
    for (int i = 0; i < nbrows; i++) {
        for (int j = 0; j < nbcolumns; j++) {
            int x = (int) (Math.random() * 100);
            System.out.println(x);
            model.setValueAt(x, i, j); // Génère des nombres aléatoires entre 1 et 99 inclus
            list_u.add(x);

        }
    }
```

FIGURE 3.12 – Procédure de génération aléatoire

3.6.2 Calcul de l'énergie (le coût) :

Cette procédure calcule le coût total associé à une solution donnée. Elle parcourt les matrices de données et de solution pour multiplier les valeurs de la matrice de données par les affectations de la solution et accumule le total du coût. Voici le code :



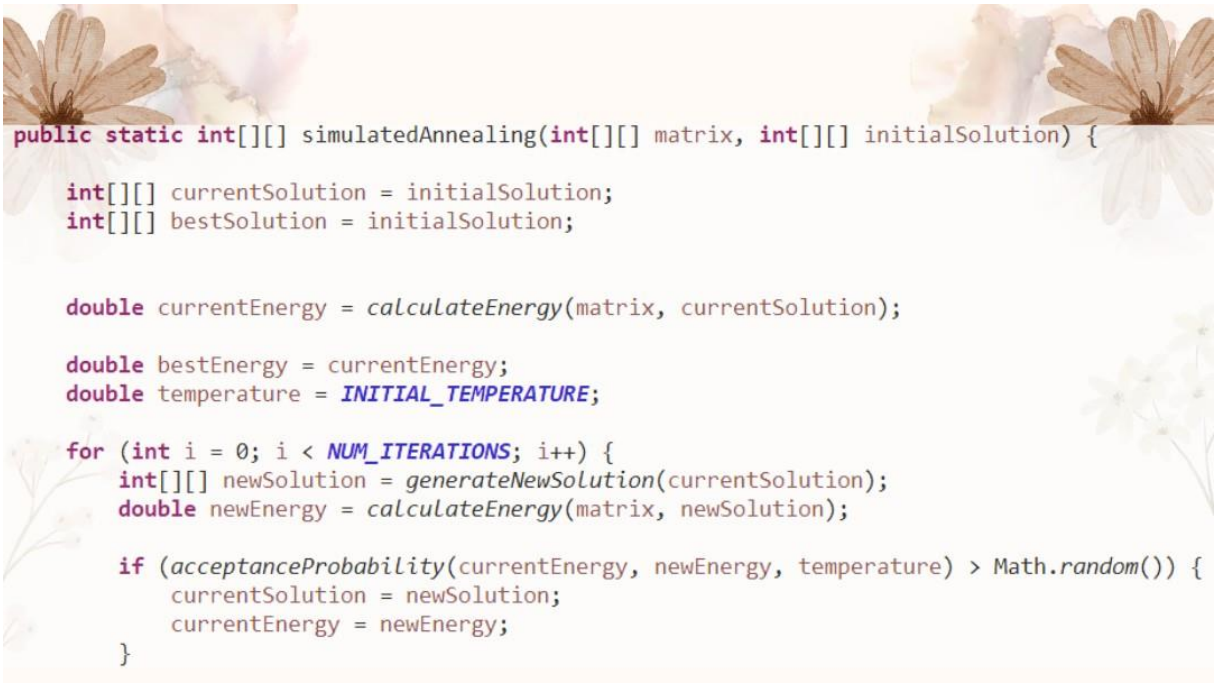
```
public static double calculateEnergy(int[][] matrix, int[][] solution) {
    int n = matrix.length;
    int m = matrix[0].length;
    double totalEnergy = 0.0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            totalEnergy += solution[i][j] * matrix[i][j];
        }
    }

    return totalEnergy;
}
```

FIGURE 3.13 – Procédure de calcul du coût total

3.6.3 Algorithme de recuit simulé

Cette procédure implémente l'algorithme de recuit simulé pour optimiser les affectations des produits aux machines. Elle utilise les températures initiales et le nombre d'itérations spécifiés par l'utilisateur pour exécuter l'algorithme. Elle utilise des boucles

et des conditions pour générer de nouvelles solutions en effectuant des mouvements aléatoires et en acceptant des solutions de qualité inférieure avec une probabilité dépendant de la température. Voici le code :



```
public static int[][] simulatedAnnealing(int[][] matrix, int[][] initialSolution) {

    int[][] currentSolution = initialSolution;
    int[][] bestSolution = initialSolution;


    double currentEnergy = calculateEnergy(matrix, currentSolution);

    double bestEnergy = currentEnergy;
    double temperature = INITIAL_TEMPERATURE;

    for (int i = 0; i < NUM_ITERATIONS; i++) {
        int[][] newSolution = generateNewSolution(currentSolution);
        double newEnergy = calculateEnergy(matrix, newSolution);

        if (acceptanceProbability(currentEnergy, newEnergy, temperature) > Math.random()) {
            currentSolution = newSolution;
            currentEnergy = newEnergy;
        }
```

FIGURE 3.14 – Procédure d'implémentation de l'algorithme de RS

3.6.4 Affichage graphique :

Cette procédure affiche graphiquement les résultats de la solution initiale et de la meilleure solution après l'exécution de l'algorithme. Elle utilise des composants graphiques tels que JFrame, JScrollPane et JTable pour afficher les matrices de solution sous forme de tableau et les visualiser de manière conviviale. Voici le code :

```
public void actionPerformed(ActionEvent e) {

    // Get the number of rows and columns
    int nbrows = Integer.valueOf(textField.getText());
    int nbcolumns = Integer.valueOf(text.getText());

    // Create a new JFrame to hold the graph
    JFrame frame = new JFrame("Graph Initiale");

    // Create a JPanel to draw the graph
    JPanel graphPanel = new JPanel() {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            int cellWidth = 70;
            int cellHeight = 100;
```

FIGURE 3.15 – Procédure d'affichage graphique

3.7 Analyse et Optimisation des Paramètres de Contrôle

L'étude et l'analyse des paramètres de contrôle dans l'algorithme de recuit simulé sont essentielles pour comprendre leur impact sur les performances et la qualité des solutions obtenues.L'algorithme de recuit simulé repose sur un ensemble restreint de paramètres, chacun ayant une signification spécifique :

- **La solution initiale :** C'est la configuration initiale du système sur laquelle l'algorithme va se baser pour effectuer les itérations et explorer l'espace des solutions.
- **La température initiale (T) :** Elle joue un rôle crucial dans le processus de recuit. Elle détermine la probabilité d'accepter des solutions de moins bonne qualité au début de l'algorithme, permettant une exploration plus large de l'espace des solutions.
- **Le nombre d'itérations (I) :** C'est le nombre total d'itérations que l'algorithme effectuera avant de s'arrêter. Plus le nombre d'itérations est élevé, plus l'algorithme aura de chances de converger vers une solution optimale.
- **Le facteur de décroissance :** C'est un coefficient qui réduit progressivement la température à chaque itération. Il contrôle la vitesse à laquelle la température diminue, ce qui impacte la manière dont l'algorithme explore l'espace des solutions.

Pour étudier l'influence de ces paramètres, il est possible de fixer certains d'entre eux à des valeurs spécifiques, par exemple :

- Fixer le facteur de décroissance à une valeur donnée, comme $q = 0,999$.
- Fixer la température initiale à une valeur spécifique, par exemple $T = 1000$.
- Fixer le nombre d'itérations à une valeur déterminée, par exemple $I = 500$.

En expérimentant avec différentes combinaisons de ces paramètres, il est possible d'observer l'effet qu'ils ont sur les performances de l'algorithme de recuit simulé et la qualité des solutions obtenues.

3.7.1 Tests sur la solution initiale

Après avoir effectué des tests sur la solution initiale en la changeant de manière aléatoire à chaque itération et en exécutant l'algorithme de recuit simulé sur la même matrice d'affectation, il est possible d'observer les variations du temps d'exécution et du coût final de la solution à chaque test.

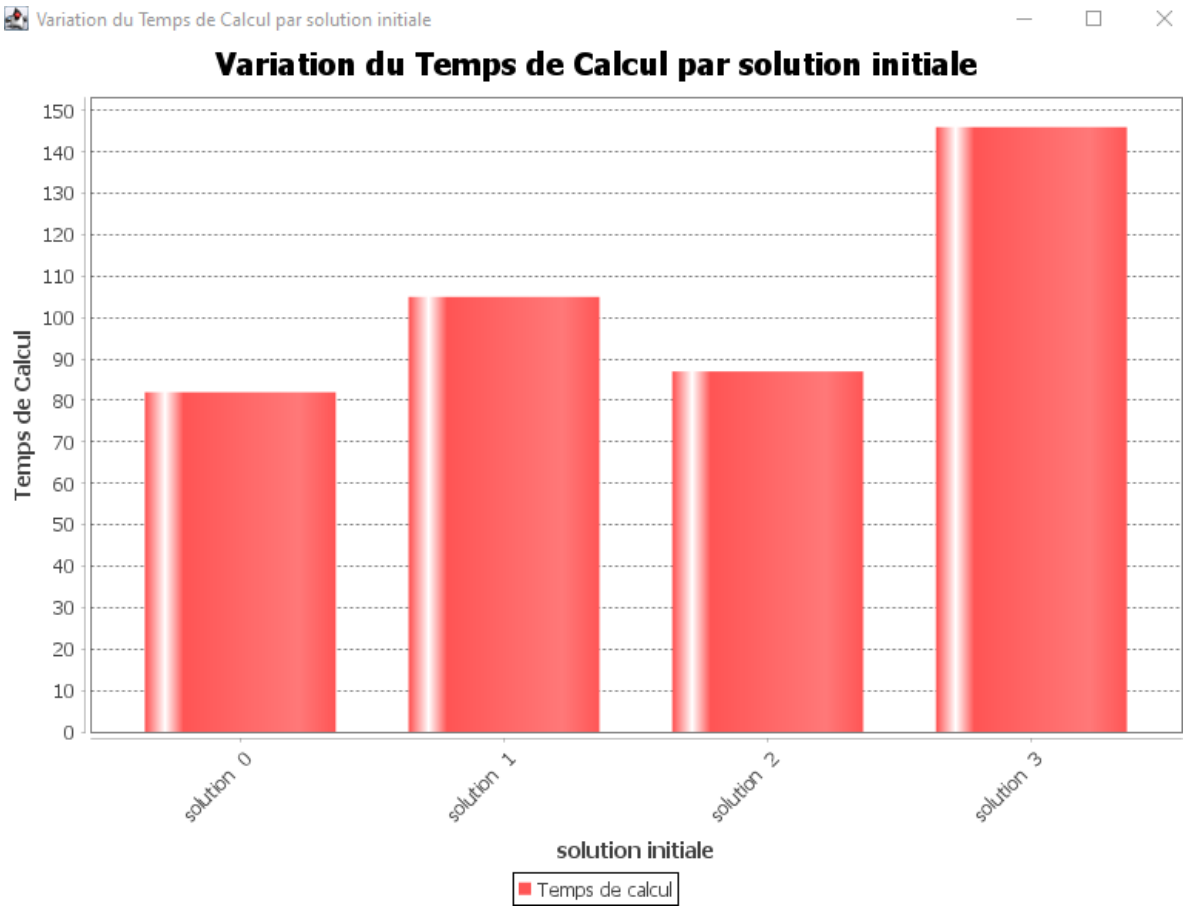


FIGURE 3.16 – Diagramme à bandes du temps de calcul par solution initiale

3.8 Conclusion

Notre application offre une solution complète basée sur l'algorithme de recuit simulé pour résoudre le problème d'affectation des produits aux machines. Elle utilise des structures de données appropriées et des procédures efficaces pour générer et optimiser les affectations, aboutissant à des solutions optimisées qui répondent aux besoins spécifiques du problème.

Conclusion générale

En conclusion, l'algorithme de recuit simulé offre une approche efficace pour résoudre le problème d'affectation de produits aux machines. Il permet d'explorer de manière heuristique l'espace des solutions, en tenant compte de l'affectation des coûts, de la génération des solutions initiales et des critères d'arrêt.

En ajustant les paramètres de l'algorithme, tels que la température initiale et le taux de décroissance, il est possible d'obtenir des résultats optimaux ou proches de l'optimalité. Cependant, il est important de noter que l'efficacité de l'algorithme dépend fortement de la qualité de l'affectation des coûts et de la génération des solutions initiales. De plus, il est crucial de choisir des critères d'arrêt appropriés pour garantir la convergence vers une solution satisfaisante. En combinant l'algorithme de recuit simulé avec d'autres techniques d'optimisation, il est possible d'améliorer davantage la qualité des solutions et d'obtenir des performances optimales dans la résolution du problème d'affectation de produits aux machines.

Bibliographie

[1] D. Orkia, “Méthodes d’optimisation,” Université Dr Moulay TAHAR-Saida, 2023.

[2] Wikipédia, “Wikipédia,” Sous licence CC-BY-SA 3.0.

[3] D. Olivier, “Traveling salesman problem (tsp) with miller-tucker-zemlin (mtz) in cplex/opl,” 24/06/2020.

[4] L. LOUKIL, “Métaheuristiques hybrides parallèles pour le q3ap sur environnements à grande échelle,” Ph.D. dissertation, Université d’Oran1-Ahmed Ben Bella, 2010.

[5] L. Lakhdar, “Métaheuristiques hybrides parallèles pour le q3ap sur environnements à grande échelle,” Ph.D. dissertation, Université d’Oran1-Ahmed Ben Bella, 2010.

[6] Y. K. . J.-C. B. Yifang Li, “Le problème du voyageur de commerce,” 16/10/2008.

[7] M. M. Mourad, “Optimisation du plan de tension et de la répartition de la puissance réactive par les techniques intelligentes hybrides améliorées,” Ph.D. dissertation, Université de Tébessa, 1955.

[8] O. Dr.Derkaoui, “Méthodes d’optimisation,” Université Dr Moulay TAHAR-Saida, 2022.

[9] “Playlist youtube,” disponible sur YouTube : https://www.youtube.com/watch?v=vr886frwE_s&list=PLBWD3LpmoU7EjjT961_MZJYun3wg2PPjn.

[10] A. B. CHILINGUIRIAN, “Résolution du problème du voyageur de commerce métaheuristique.”

[11] H. HACENE, “Etude comparative des méthodes heuristiques d’optimisation combinatoire,” Ph.D. dissertation.

[12] H. Boukef, “Algorithme relatif au fonctionnement général du recuit simulé.”

[13] M. L. Meklid, “Recuit simulé - complex systems and ai,” 10 février 2016.

[14] W. Segretier, “Metaheuristiques et optimisation combinatoire,” *LAboratoire de Mathématiques, Informatique et Applications (LAMIA)*, vol. 1, p. 33, février 2019.

[15] [Online]. Available : https://ralph.blog.imixs.com/2022/09/15/how-to-install-a-fast-small-eclipse-ide

[16] [Online]. Available : https://www.softwaretestinghelp.com/java/java-development-using-eclipse-ide

[17] A. BAYOU and A. BENSEFIA, "Un algorithme hybride (aco-2opt) pour la résolution du problème de voyage de commerce," Ph.D. dissertation, Université de Bordj Bou Arreridj Faculty of Mathematics and Computer Science, 2021.

[18] Y. K. . J.-C. B. Yifang Li, "Le problème du sac à dos," 11/07/2008 school = l'Université de Tours ,.

[19] C. Mancel, "modélisation et résolution de problèmes d'optimisation combinatoire issus d'applications spatiales," Ph.D. dissertation, INSA de Toulouse, 2004.

[20] B. A/MALEK, "Optimisation combinatoire (oc)," 2020/2021, université de Tiaret.

[21] S. BRAHIMI, M. E. A. MIMOUNE *et al.*, "Les problèmes d'ordonnancement multi-objectifs dans un système de production," Ph.D. dissertation, UNIVERSITY of M'SILA, 2022.

[22] M. de l'Éducation nationale et de la jeunesse. (2019, septembre) Ministère de l'Éducation nationale et de la jeunesse. eduscol.education.fr. [Online]. Available : https://eduscol.education.fr/

[23] H. H. Hoos and T. Stützle, *Stochastic local search : Foundations and applications*. Elsevier, 2004.

[24] M. Bellalouna, "Problèmes d'optimisation combinatoires probabilistes," Ph.D. dissertation, Ecole Nationale des Ponts et Chaussées, 1993.

[25] Y. Li, Y. Kergosien, and J.-C. Billaut, "Le problème du voyageur de commerce," *Niveau intermédiaire*, 10 2008.

[26] P. Loubiere, "Amélioration des métaheuristiques d'optimisation à l'aide de l'analyse de sensibilité," Ph.D. dissertation, Paris Est, 2016.

[27] M. M. Mourad, "Optimisation du plan de tension et de la répartition de la puissance réactive par les techniques intelligentes hybrides améliorées," Ph.D. dissertation, Université de Tébessa, 1955.

[28] H. Hachimi, "Hybridations d'algorithmes métaheuristiques en optimisation globale et leurs applications," Ph.D. dissertation, Rouen, INSA, 2013.

[29] P. Hahn, T. Grant, and N. Hall, "A branch-and-bound algorithm for the quadratic assignment problem based on the hungarian method," *European Journal of Operational Research*, vol. 108, no. 3, pp. 629–640, 1998.

[30] K. Aardal, A. Hipolito, C. Van Hoesel, B. Jansen, C. Roos, and T. Terlaky, "A branch-and-cut algorithm for the frequency assignment problem," *Research Memorandum*, vol. 96, no. 011, pp. 3–7, 1996.

[31] R. E. Burkard, E. Cela, P. M. Pardalos, and L. S. Pitsoulis, *The quadratic assignment problem*. Springer, 1998.

[32] P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo, "Incremental local search in ant colony optimization : Why it fails for the quadratic assignment problem," in *Ant Colony Optimization and Swarm Intelligence : 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006. Proceedings 5*. Springer, 2006, pp. 156–166.

[33] A. Misevičius, "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, no. 4, pp. 497–514, 2003.

[34] G. C. Onwubolu, B. Babu, G. C. Onwubolu, and A. Sharma, "Particle swarm optimization for the assignment of facilities to locations," *New Optimization Techniques in Engineering*, pp. 567–584, 2004.

[35] W. Zhu, J. Curry, and A. Marquez, "Simd tabu search for the quadratic assignment problem with graphics hardware acceleration," *International Journal of Production Research*, vol. 48, no. 4, pp. 1035–1047, 2010.

[36] G. A. Azim, "Neural networks for solving quadratic assignment problems," *Neural Information Processing-Letters and Reviews*, vol. 10, no. 3, pp. 27–34, 2006.

[37] M. L. Meklid, "La réalisation d'un algorithme génétique pour la résolution du problème d'affectation de produits aux machines," 2015.

[38] D. N. KHERICI. Optimisation combinatoire.

[39] A. BENDAHMANE, "Le recuit simulé," *Université des Sciences et de la Technologie d'Oran-Mohamed Boudiaf*, 2011.

[40] ——, "Le recuit simulé," *Université des Sciences et de la Technologie d'Oran-Mohamed Boudiaf*, 2011.

[41] R. Sessa and J. Esterkin, "Définissez votre architecture logicielle grâce aux standards reconnus," *01/03/2022*.

[42] Android Developers. Couche de données. Consulté le 13 juin 2023. [Online]. Available : https://developer.android.com/topic/architecture/data-layer?hl=fr

[43] H. HACENE, "Etude comparative des méthodes heuristiques d'optimisation combinatoire," Ph.D. dissertation.

[44] [Online]. Available : https://www.eclipse.org

[45] [Online]. Available : https://www.oracle.com/java