

# CHAPITRE 1

## Le Problème D'optimisation Combinatoire

### 1. Introduction:

Le domaine de l'optimisation combinatoire est un domaine très important, il est situé au carrefour de la recherche opérationnelle, mathématique et informatique. Son importance s'explique par la grande difficulté posée par les problèmes d'optimisation d'une part, et par le nombre important des applications pratiques pouvant être formulées sous forme de problèmes d'optimisation combinatoires d'autre part. L'optimisation combinatoire consiste à parcourir l'espace de recherche (l'ensemble de combinaisons pouvant être prises par les variables du problème) afin d'en extraire une solution optimale parmi un ensemble fini de solutions, d'une taille souvent très grande telle que son énumération exhaustive est une tâche fastidieuse. En fait, un problème d'optimisation combinatoire se ramène à résoudre une de ses instances par un procédé algorithmique permettant la maximisation (en cas de problème de maximisation) ou la minimisation (en cas de problème de maximisation) d'une (ou de plusieurs) fonction(s) objectif(s) en respectant certaines contraintes rendant infaisable une partie de l'espace de recherche. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes dites **NP-difficiles** et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données. Nous présentons dans ce chapitre l'optimisation combinatoire, problème d'optimisation, le voisinage, la théorie de la complexité, quelque problème d'optimisation et la méthode de résolution d'un problème.

### 2. Définition le problème d'optimisation combinatoire :

Un problème d'optimisation combinatoire (OC) consiste à déterminer un plus grand (petit) élément dans un ensemble fini value. En d'autres termes, étant donnés une famille  $F$  de sous-ensembles d'un ensemble ni  $E = \{e_1, \dots, e_n\}$  et un système de poids  $w = (w(e_1), \dots, w(e_n))$  associé aux éléments de  $E$ , un problème d'optimisation consiste à trouver un ensemble  $F \in F$  de poids  $w(F) = \sum_{e \in F} W(e)$  maximum (ou minimum), i.e.

$$\max \text{ ou } \min \{w(F) \mid F \in F\}.$$

La famille  $F$  représente donc les solutions du problème. Elle peut correspondre à un ensemble de très grande taille que l'on ne connaît que par des descriptions ou des propriétés théoriques qui ne permettent pas facilement son énumération.[1].

### 3. Complexité d'un problème:

La complexité d'un algorithme c'est le temps et l'espace mémoire nécessaires pour son exécution. Elle est calculée en fonction du nombre  $N$  des données appelées aussi taille du problème [4]. Dans le cas le plus défavorable, la théorie de complexité permet de majorer le nombre d'opérations nécessaires par une fonction de la taille  $N$  du problème posé. La complexité d'un algorithme est notée  $F(N)$ , s'il  $\exists$  une constante  $C$  et un entier  $A$  tels que le nombre d'instructions élémentaires  $I(N)$  vérifie  $I(N) \leq CF(N)$  pour tout  $N \geq A$ .

Si  $F(N)$  est un polynôme alors l'algorithme est dit polynomial [6]. Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisation. Elle classe les problèmes selon leurs complexités en deux classes principales: la classe **P** (Polynomial time) et la classe **NP** (Non deterministic Polynomial time). En outre, elle partage les problèmes de la classe **NP** en deux sous classes: **NP-Complet** et **NP-Difficile**.

#### 3.1. Classification des problèmes d'optimisation :

- **Le problème de classe P :** L'appartenance d'un problème quelconque à cette classe nécessite qu'il soit un problème de décision et ayant au moins un algorithme polynomial comme méthode de résolution, il est nommé facile ou de classe **P**. La vérification de l'existence d'un tel algorithme est obligatoire pour montrer l'appartenance du problème à cette classe. [6]
- **Le problème de classe NP :** Le problème de cette classe ne peut pas être résolu dans un temps polynomial, il est dit **NP** difficile. Les méthodes de résolution utilisées sont les heuristiques qui permettent de trouver une solution optimale mais non démontrables. Les problèmes de classe **P** peuvent être résolus par les algorithmes ordinaires ou exacts qui sont inclus dans la famille des heuristiques, par la suite la classe **P** est inclus dans la classe **NP**. [6]
- **Le problème de classe NP-Complet :**

Un problème de décision **P** est dit **NP-complet** s'il appartient à la classe **NP** et si pour tout problème **P'** de **NP**, on a les propriétés suivantes :

  - Il existe une application polynomiale qui transforme toute instance  $I'$  de **P'** en une instance  $I$  de **A**.
  - **P'** admet une réponse «oui » pour l'instance  $I'$ , si et seulement si **P** admet une réponse oui pour l'instance  $I$ .

Une propriété générale des problèmes **NP** complet : Si un seul problème **NP**-complet est polynomial, alors tous les problèmes **NP**-complet sont polynomiaux. Si le problème est **NP**-difficile, il n'existe pas d'algorithme polynomial le résolvant [7].

➤ **Le problème de classe NP-Difficiles :**

La classe de problèmes **NP**-Difficiles englobe les problèmes de décision et les problèmes d'optimisation. Les problèmes **NP**-Difficiles sont aussi difficiles que les problèmes **NP** Complets. Si un problème de décision associé à un problème d'optimisation **P** est **NP**-Complet alors **P** est un **NP**-Difficile. Par conséquent, afin de prouver qu'un problème d'optimisation est **NP**-Difficile, il suffit de montrer que le problème de décision associé à **P** est **NP**-Complet. Il est à noter que jusqu'à maintenant, aucun algorithme polynomial n'est connu pour résoudre ce type de problèmes (i.e. **NP**- Difficiles). [8]

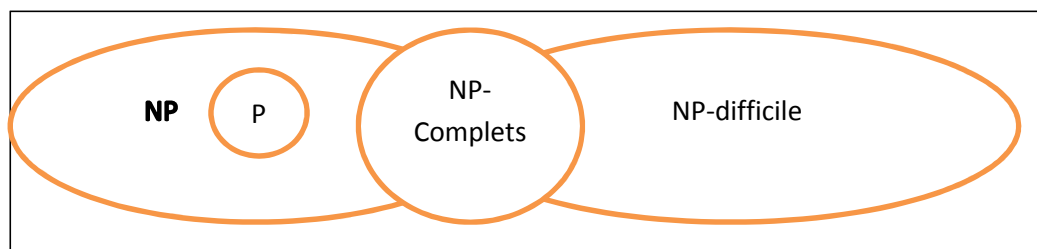
**3.2. Les problèmes NP-Complets versus les problèmes NP-Difficiles :**

Les problèmes **NP**-Complets et les problèmes **NP**-Difficiles sont deux types de problèmes difficiles. Aucun algorithme polynomial permettant leur résolution n'a été trouvé. En fait, les algorithmes proposés dans la littérature sont de complexité exponentielle ou pire qu'exponentielle. La différence entre les problèmes de ces deux types de problèmes réside dans le type de la solution attendue. Les problèmes **NP**-Complets sont des problèmes décisionnels dont la réponse attendue pourra être « *oui ou non* ». Comme l'exemple du problème du cycle Hamiltonien qui consiste à répondre à la question: Est-ce qu'il existe un cycle permettant de parcourir tous les sommets d'un graphe non orienté en passant par chacun une seule fois ? Ici on s'attend à une des deux réponses: oui il existe **ou** non il n'existe pas. Par contre, les problèmes **NP**-Difficiles concernent beaucoup plus les problèmes d'optimisation ou on cherche la solution optimale. Comme le cas du problème du voyageur de commerce dont on cherche le plus court chemin permettant de parcourir un ensemble de sommets (villes) en passant par chacun une seule fois.[8]

**3.3. La relation entre les problèmes P et NP :**

La question « **P=NP** ? » est une des questions les plus importantes qui n'ont pas encore été résolues en informatique théorique. En fait, la réponse à cette question a construit un champ de recherche ouvert. La réponse à la question « **P=NP** ? » par « oui », revient à montrer

que tous les problèmes de la classe **NP** sont dans la classe **P**. Autrement dit, la réponse à cette question revient à répondre à la question: Peut-on trouver en temps polynomial ce qu'on peut prouver en temps polynomial ? Cependant, aucune démonstration n'a été faite pour prouver que  $NP \subseteq P$ , ni que  $NP \not\subseteq P$ . La relation évidente c'est que  $P \subseteq NP$ . En effet, un problème qui peut être résolu en un temps polynomial par un algorithme déterministe, pourra aussi être résolu par un algorithme non déterministe. Ce qui est admis et connu jusqu'à maintenant, c'est que  $P \neq NP$ . Néanmoins, aucune preuve n'a encore été prouvée jusqu'à ce jour. Si un jour on arrivera à trouver un algorithme polynomial permettant la résolution d'un problème **NP-Complet**, on pourra résoudre tous les problèmes **NP-Complets** en temps polynomial (voir section 3.2.1). Cook a prouvé dans que tous les problèmes de la classe **NP** sont réductibles au problème de la satisfiabilité d'une formule logique, cela veut dire que si quelqu'un trouvera un algorithme polynomial pour le problème de SAT, alors la question «  $P = NP$  ? » sera résolu! Selon la figure 1.1, les problèmes **NP** faciles à résoudre peuvent construire des problèmes de la classe **P**. Les autres problèmes, i.e. les plus difficiles à résoudre, peuvent être partagés en deux autres groupes: le groupe des problèmes **NP-Complets** et le groupe des problèmes dont le statut est indéterminé car ces problèmes n'ont pas été prouvés comme appartenant à la classe **P** ni à la classe **NP-Complet**, comme le cas du problème d'isomorphisme de deux graphes. En effet, il semblait que ce problème appartient à la classe **P**. Néanmoins, jusqu'à ce jour aucune preuve n'a été montrée. [8]



**Figure 1.1** : la relation entre les différentes classes du problème.

#### 4. Exemples des problèmes classiques d'optimisation combinatoire :

Un problème d'optimisation consiste à chercher une instanciation d'un ensemble de variables soumises à des contraintes, de façon à maximiser ou minimiser un critère. Lorsque les domaines de valeurs des variables sont discrets, on parle alors de problèmes d'optimisation combinatoire. Nous présentons rapidement ici quatre problèmes classiques d'optimisation combinatoire : le problème du sac-à-dos, le problème d'affectation, le problème du voyageur de commerce, le problème d'ordonnancement, le problème de couplage maximal et le

problème de flot maximal. [2]

#### 4.1. Le problème du sac-à-dos :

Le problème de sac-à-dos est très simple. On utilise pour chaque objet  $i \in \{1, \dots, n\}$ , une variable entière  $x_i$  correspondant au nombre de fois où l'objet  $i$  est choisi. Le problème du sac-à-dos est donc équivalent au programme en nombres entiers suivant :

$$\begin{aligned} \max \quad & \sum_{i=1}^n c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n a_i x_i \leq b \\ & p_i \leq x_i \leq q_i, \text{ pour } i = 1, \dots, n. \\ & x_i \in \mathbb{N}, \text{ pour } i = 1, \dots, n. \end{aligned}$$

Ce problème est NP-complet alors qu'il a une unique contrainte [1]

#### 4.2. Le problème d'affectation:

Le problème d'affectation consiste à établir des liens entre les éléments de deux ensembles distincts, de façon à minimiser un coût et en respectant des contraintes d'unicité de lien pour chaque élément. On considère  $m$  tâches et  $n$  agents, avec  $n \geq m$ . Pour tout couple  $(i, j)$  ( $i = 1$  à  $m$ ,  $j = 1$  à  $n$ ), l'affectation de la tâche  $i$  à  $j$  entraîne un coût de réalisation noté  $c_{i,j}$  ( $c_{i,j} \geq 0$ ). Chaque tâche doit être réalisée exactement une fois et chaque agent peut réaliser au plus une tâche. Le problème consiste à affecter les tâches aux agents, de façon à minimiser le coût total de réalisation et en respectant les contraintes de réalisation des tâches et de disponibilité des agents. À tout couple tâche/agent  $(i, j)$ , on associe une variable d'affectation,  $x_{i,j}$ , binaire, qui prend la valeur 1 si la tâche  $i$  est affectée à l'agent  $j$  et 0 sinon. Le coût total de réalisation des tâches s'exprime alors par la somme :  $\sum_{i=1}^m \sum_{j=1}^n c_{i,j} \cdot x_{i,j}$ . Le nombre d'agents réalisant la tâche  $i$  est donné par :  $\sum_{j=1}^n x_{i,j}$  pour tout  $i = 1$  à  $m$  et le nombre de tâches réalisées par l'agent  $j$  est donné par :  $\sum_{i=1}^m x_{i,j}$ , pour tout  $j = 1$  à  $n$ . On peut donc modéliser le problème d'affectation sous la forme :

$$\begin{aligned} \text{S.C.} \quad & \sum_{j=1}^n x_{i,j} = 1 & \forall i = 1..m \\ & \sum_{i=1}^m x_{i,j} \leq 1 & \forall j = 1..n \\ & x_{i,j} \in \{0, 1\} & \forall i = 1..m, \forall j = 1..n \end{aligned}$$

Les contraintes de ce problème se retrouvent dans de nombreuses applications mettant en jeu des problèmes d'allocation de ressources. Elles sont généralement appelées "contraintes D'affectation". [2]

#### 4.3. Le problème de couplage maximal :

En théorie des graphes, on peut se ramener à un "problème de couplage dans un graphe biparti". On dit d'un graphe  $G$  qu'il est biparti si l'on peut diviser les sommets en deux ensembles  $X_1$  et  $X_2$  de telle sorte que toutes les arêtes dans le graphe joignent un sommet de  $X_1$  à un sommet de  $X_2$ . Un "couplage" dans un graphe biparti est un ensemble d'arêtes qui n'ont, 2 à 2, aucun sommet commun dans  $G$ . En associant  $X_1$  à l'ensemble des tâches, de cardinalité  $m$  et  $X_2$  à l'ensemble des agents, de cardinalité  $n$ , une arête  $(i, j)$  dans le graphe  $G$  (avec  $i \in X_1$  et  $j \in X_2$ ) représente la possibilité d'affecter la tâche  $i$  à l'agent  $j$  ; on associe le poids  $C_{ij}$  à chaque arête  $(i, j)$  de  $G$ . Le poids d'un couplage étant défini comme la somme des poids de ses arêtes, le problème d'affectation revient alors à chercher un couplage de cardinalité  $m$  de poids minimal dans le graphe  $G$ . Le cas particulier où  $X_1$  et  $X_2$  sont de même cardinalité (correspondant au cas  $n = m$  pour le problème d'affectation) est fréquemment étudié ; on s'intéresse alors à la recherche d'un couplage de cardinalité maximale. Si on considère des ensembles  $X_1$  et  $X_2$  de cardinalité  $n$  et s'il existe  $n^2$  arêtes dans le graphe  $G$  (le graphe biparti est complet), alors le couplage maximal est de cardinalité  $n$  et il est appelé "couplage parfait". On peut étendre ce problème à celui de la recherche d'un couplage maximal de poids minimal dans  $G$ . [2]

#### 4. 4. Le problème de flot maximal :

Le problème d'affectation, ou de couplage dans un graphe biparti, peut être modélisé comme un problème de flot maximal à coût minimal dans lequel les capacités des arcs sont toutes égales à 1. C'est un problème classique de la théorie des graphes qui revient à chercher à faire passer un débit maximal à travers un réseau, pour un moindre coût. Des algorithmes simples et efficaces existent pour résoudre ce problème ; en particulier l'algorithme de Busaker *et* Gowen qui part d'un flot nul et qui l'augmente progressivement par recherche de "chaînes augmentantes" (*i.e.*, chemins sur les arcs desquels on peut systématiquement augmenter le flot), de coût minimal. [2]

#### 4.5. Le problème de coloration :

Soit  $S \subseteq IN$ . Une coloration des sommets d'un graphe  $G = (V, E)$  est une fonction  $r : V \rightarrow S$  telle que  $r(u) \neq r(v)$  pour tout couple de sommets adjacents  $u, v$ . Les éléments de l'ensemble  $S$  sont appelés les couleurs disponibles. Une  $k$ -coloration est une coloration  $c : V \rightarrow \{1, \dots, k\}$ .

Un graphe est dit  $k$ -coloriable s'il possède une  $k$ -coloration. Tester si un graphe est  $k$ -coloriable est un problème **NP-complet** si  $k \geq 3$ , et il est polynomial si  $k = 2$ . En eet, si  $k = 2$ , il suffit de tester si le graphe est biparti, c'est-à-dire s'il ne contient pas des cycles impair. Ce qui peut se faire par un simple parcours de graphe. Soit  $G = (V, E)$  un graphe non-orienté. Le problème de coloration consiste à déterminer le plus petit  $k$  tel que  $G$  soit  $k$ -coloriable. [1].

#### 4.6. Le problème du voyageur de commerce (PVC).

Le "problème du voyageur de commerce", ou pvc (en anglais Traveling Salesman Problem) est un problème il s'agit de trouver le chemin le plus court reliant  $n$  villes données, chaque ville ne devant être visitée qu'une et une seule fois, et revenir à la ville de départ. La difficulté du problème vient de l'explosion combinatoire du nombre de chemins à explorer lorsque l'on accroît le nombre de villes à visiter. Plus formellement, ce problème peut être modélisé comme suit : Etant donné un graphe non orienté complet  $G = (S, A)$ , et une fonction de distance  $d : A \rightarrow \mathbb{R}^+$ , on cherche à déterminer un cycle Hamiltonien 2 de distance totale minimale. Pour ce problème, trouver s'il existe un chemin Hamiltonien est un problème **NP-complet**, la recherche du plus court chemin Hamiltonien est un problème **NP-difficile**.

Le problème PVC est alors modélisé par.

$$\text{minimiser } \sum_{i=1}^{n-1} (D(V_i, V_{i+1})) + D(V_n, V_1)$$

Ou  $D(V_i, V_{i+1})$  est la distance entre les deux villes  $V_i$  et  $V_{i+1}$ . [3]

### 5. Les méthodes de résolution d'un problème d'optimisation :

Il existe deux méthodes pour la résolution du problème d'optimisation : méthodes exactes et méthodes approchées, La figure 1.2. met en parallèle les méthodes représentatives développée En recherche Opérationnelle.

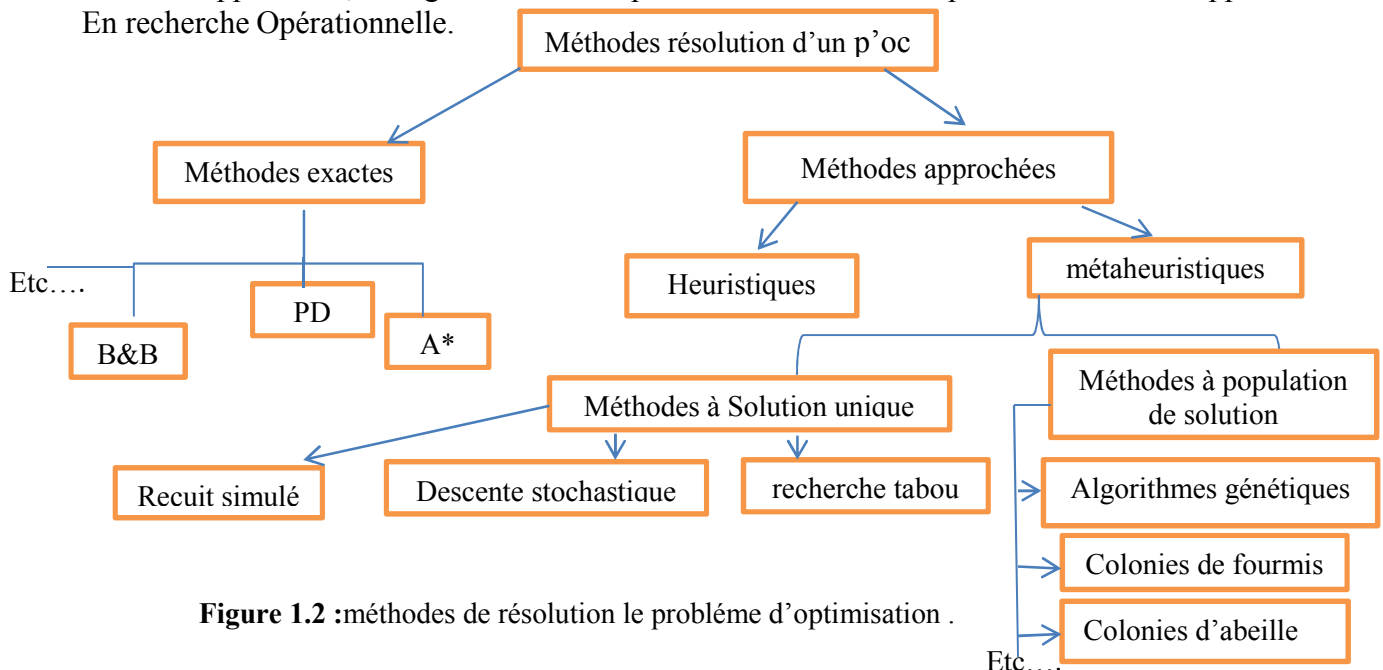


Figure 1.2 :méthodes de résolution le problème d'optimisation .

### **5.1. Les méthodes exactes:**

Les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles. De ce fait, la complexité de ce type d'algorithme croît exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions objectifs et /ou critères. Il existe de nombreux algorithmes exacts y compris la programmation dynamique, les algorithmes de séparation et évaluation (Branch and Bound).....etc. [8]

### **5.2. Les méthodes approchées :**

Une méthode approchée ou heuristique est un algorithme d'optimisation qui a pour but de trouver une solution raisonnable de la fonction objectif, mais sans garantir l'optimalité. Le principal avantage de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, bien ou mal formulés, avec ou sans contrainte. En particulier, elles ne nécessitent pas une modélisation mathématique du problème. elles semblent être tout à fait adaptées à l'optimisation de systèmes de production et c'est donc ce type de méthode qui sera utilisé dans la suite de ce travail. Le problème de transport [9]

#### **5.2.1. Les Heuristiques :**

Une heuristique (méthode heuristique) est un algorithme qui fournit rapidement en temps Polynomial une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile.[10]

#### **5.2.2. Les Métaheuristiques :**

Les méthodes dites métaheuristiques sont des méthodes générales, des heuristiques polyvalentes applicables sur une grande gamme de problèmes. Elles peuvent construire une alternative aux méthodes heuristiques lorsqu'on ne connaît pas l'heuristique spécifique à un problème donné. [8] L'ensemble des métaheuristiques proposées dans la littérature sont partagées en deux classes: des métaheuristiques à base de population de solutions et des métaheuristiques à base de solution unique.

##### **5.2.2.1. Les Métaheuristiques à base de population de solutions :**

Les métaheuristiques à base de population de solutions débutent la recherche avec une panoplie de solutions. Elles s'appliquent sur un ensemble de solutions afin d'en extraire la meilleure (l'optimum global) qui représentera la solution du problème traité. L'idée d'utiliser



un ensemble de solutions au lieu d'une seule solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité. Une grande variété de méthodes basées sur une population de solutions a été proposée dans la littérature, commençant par les algorithmes évolutionnaires, passant par les algorithmes génétiques et arrivant aux algorithmes à base d'intelligence par essaims (l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou, l'algorithme d'optimisation par coucou...) qui ont connus une investigation remarquable ces deux dernières décennies.[8]

#### **5.2.2.2. Les Métaheuristiques à base de solution unique :**

Les métaheuristiques à base de solution unique débutent la recherche avec une seule solution initiale. Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations. Le voisinage de la solution s englobe l'ensemble des modifications qui peuvent être effectuées sur la solution elle-même. La qualité de la solution finale dépend particulièrement des modifications effectuées par les opérateurs de voisinages. En effet, les mauvaises transformations de la solution initiale mènent la recherche vers la vallée de l'optimum local d'un voisinage donné (peut être un mauvais voisinage) ce qui bloque la recherche en fournissant une solution de qualité insuffisante. De nombreuses méthodes à base de solution unique ont été proposées dans la littérature. Parmi lesquelles: la descente, le recuit simulé, la recherche tabou, la recherche à voisinage variable (VNS: Variable Neighbourhood Search), la recherche locale réitérée (ILS: Iterated Local Search), la recherche locale guidée (GLS: Guided Local Search)...etc.[8]

### **6. Notion de voisinage :**

Les heuristiques et les métaheuristiques manipulent une ou plusieurs solutions, et mettent souvent en œuvre des stratégies de mouvement d'une solution vers une autre. C'est en particulier le cas avec les techniques basées sur la recherche locale. Cette notion de mouvement d'une solution vers une autre est liée à la définition du voisinage de la solution considérée. Le voisinage d'une solution s correspond à l'ensemble des solutions accessibles depuis s à l'aide d'un mouvement élémentaire. Plus formellement, soit  $\langle S, f \rangle$  l'instance du problème d'optimisation P traité, et soit  $d : S \times S \rightarrow \mathbb{R}^+$  une fonction qui calcule la distance

entre deux solutions de  $S$ . Un voisinage d'une solution  $s \in S$ , est un ensemble  $V(s) \subseteq S$  des solutions "proches" de  $s$ , c'est-à-dire à une distance inférieure à  $\varepsilon$  donné :

$$V(s) = \{y \in S \mid d(s, y) \leq \varepsilon, \varepsilon \in \mathbb{R}^+\} \quad (1.1)$$

Illustrons cette notion de voisinage sur le problème du voyageur de commerce. L'ensemble des solutions  $S$  de ce problème est constitué de toutes les permutations possibles des villes. Un voisinage classique pour ce problème appelé **2-OPT** consiste à supprimer deux arêtes de la tournée courante et en ajouter deux autres arêtes. [11]

Exemple de voisinage :

Pour le problème du voyageur de commerce il y'a

- Le voisinage par *2-OPT*
- Le voisinage par *3-OPT*
- Le voisinage par *K-OPT*

### 6.1. Optimum local :

Une solution  $s' \in X$  est un optimum local si :

$$\forall s \in V(s) = \begin{cases} f(s') \leq f(s) & \text{dans le cas d'un problème de minimisation} \\ f(s') \geq f(s) & \text{dans le cas d'un problème de maximisation} \end{cases}$$

Avec  $V(s)$  l'ensemble des solutions voisines de  $s$ .

### 6.2. Optimum global :

Une solution  $s' \in X$  est un optimum global si :

$$\forall s \in X = \begin{cases} f(s') \leq f(s) & \text{dans le cas d'un problème de minimisation} \\ f(s') \geq f(s) & \text{dans le cas d'un problème de maximisation} \end{cases}$$

## 7. Conclusion :

Dans ce qui suit on va considérer des problèmes d'optimisation combinatoire : le problème de transport pour lequel on a proposé des nouvelles heuristiques qui ne sont qu'une généralisation de la méthode de coin-Nord-Ouest, pour déterminer une solution admissible. Puis on considère le problème d'affectation dont la méthode de solution exacte sera utiliser pour déterminer une évaluation par défaut pour la fonction objective du problème du voyageur de commerce pour lequel on l'utilise pour construire une méthode pour séparation et évaluation (B&B) pour déterminer une solution exacte. Pour le PVC de grande taille des métaheuristiques comme la méthode taboue et la méthode recuit simulé ont aussi été établi pour déterminer des solutions approchées.