

## CHAPITRE II

# LES METHODES DE RESOLUTION D'UN PROBLEME D'ORDONNANCEMENT

### 1. Introduction :

Pour résoudre un problème d'ordonnancement, il ne suffit pas de prouver l'existence d'une solution. Il faut également la construire. Il est évidemment plus difficile de construire une solution que de prouver son existence.

La nature complexe des problèmes d'ordonnancement en général, est due principalement au fait qu'ils sont des problèmes d'optimisation hautement combinatoire. Au fait, dans un problème d'optimisation hautement combinatoire, on a à extraire d'un ensemble fini, un "meilleur" élément selon le (s) critère(s) préfixé(s).

Quoique fini, l'ensemble objet de l'étude comporte en général un nombre astronomique d'éléments à comparer [SAK84].

Il n'existe pas des méthodes universelles permettant de résoudre efficacement tous Les cas, beaucoup d'entre eux peuvent prendre un temps considérable pour être résolus.

Pour résoudre un problème donné, plusieurs méthodes de résolutions peuvent être mises à notre disposition, il est alors intéressant d'évaluer le temps et/ou l'espace requis pour exécuter l'algorithme correspondant à la stratégie utilisée. Cette étape est celle de l'analyse des algorithmes.

Evaluer un algorithme c'est déterminer son efficacité en évaluant sa complexité [CAR88].

### 2. Complexité des algorithmes

Un même problème peut généralement être résolu par plusieurs algorithmes, donc il faut comparer entre ces algorithmes, cette comparaison se base sur le temps de calcul et sur l'espace mémoire requis par l'algorithme.

## 2.1. Définition

On appelle complexité en temps d'un algorithme dans le pire cas, la fonction  $f(n)$  qui donne une borne supérieure du nombre d'opérations élémentaires effectuées par l'algorithme lorsque la taille de l'entrée est  $n$ .

## 2.2. Un algorithme polynomiale

Est un algorithme dont le nombre d'opérations élémentaire nécessaires pour résoudre un exemple de taille  $n$  est une fonction polynomiale en  $n$ .

## 2.3. Algorithme efficace

Schématiquement, on dit qu'un algorithme est efficace (qu'il est <bon>) si le nombre des opérations nécessaires pour résoudre un problème est borné par une fonction polynomiale d'un paramètre caractérisant la taille de ce problème.

Il peut se trouver que, pour un problème donné, toutes les méthodes de résolution connue jusqu'à date ont des temps d'exécution non polynomiaux.

Le problème dans ce cas est de savoir si cela est dû à notre incapacité de trouver une solution polynomiale ou alors que le problème est intrinsèquement difficile à résoudre.

Enfin, le choix de la méthode de résolution à mettre en œuvre dépendra souvent de la complexité du problème.

## 3. Complexité des problèmes

Distinction entre problème de décision et problème d'optimisation

**1. Un problème de décision** est un problème pour lequel une solution est soit "oui" soit "non".

**2. Un problème d'optimisation** est un problème pour lequel on doit chercher à déterminer une solution qui optimise un critère.

A chaque problème d'optimisation on peut associer un problème de décision.

### 3. Réduction polynomiale

Soient  $\Pi$  et  $\Pi'$  deux problèmes de décision.  $\Pi'$  se réduit polynomialement à  $\Pi$  et on note  $\Pi' \leq \Pi$ , si  $\Pi'$  est polynomial ou s'il existe un algorithme polynomial qui construit, à partir d'une donnée  $D'$  de  $\Pi'$ , une donnée  $D$  de  $\Pi$  telle que, la réponse pour  $D'$  soit oui si et seulement si la réponse pour  $D$  est oui [CAR 88].

**Remarque:**

Si  $\Pi' \alpha \Pi$  on dira que  $\Pi$  est plus difficile que  $\Pi'$  et que  $\Pi'$  est plus facile que  $\Pi$  [CAR 88].

Les problèmes  $\Pi'$  et  $\Pi$  sont équivalents si :  $\Pi' \alpha \Pi$  et  $\Pi \alpha \Pi'$

**4. La classe NP**

La classe *NP* est la classe des problèmes de décision qui peuvent être résolus par une machine de Turing non déterministe en temps polynomial.

On distingue trois sous classes :

**a) La classe P**

Un problème appartient à la classe *P* s'il existe un algorithme polynomial pour le résoudre.

Tout problème de la classe *P* est solvable par une machine de Turing déterministe en temps polynomial.

La classe *P* contient les problèmes les plus faciles de *NP*.

**b) La classe des problèmes NP-complets**

Un problème *D* de décision est dit *NP-complet* si  $D \in NP$  et  $\Pi \alpha D$  quelque soit  $\Pi \in NP$ . La classe des problèmes *NP-complets* contient les problèmes les plus difficiles de *NP*.

**c) La classe des problèmes ouverts**

C'est la classe de problèmes de statut indéterminé [CAR88].

**Remarque:**

En réalité le concept de *NP-complétude* s'applique aux problèmes de décision. Ainsi chaque fois qu'on dit qu'un problème d'ordonnancement est *NP-complet*, ceci sous-entend le problème de décision qui lui est associé. Par conséquent, un algorithme polynomial soit qu'il existe pour les deux, soit qu'il n'existe pour aucun.

**5. Les problèmes NP-difficile**

Un problème d'optimisation est dit *NP-difficile* si le problème de décision associé est *NP-complet*.

Il existe un certain nombre de résultats dans la littérature qui montrent les liens, sous forme d'arbres de réduction entre différents problèmes d'ordonnancement. Arbre de réduction en fonction des critères d'optimisation.

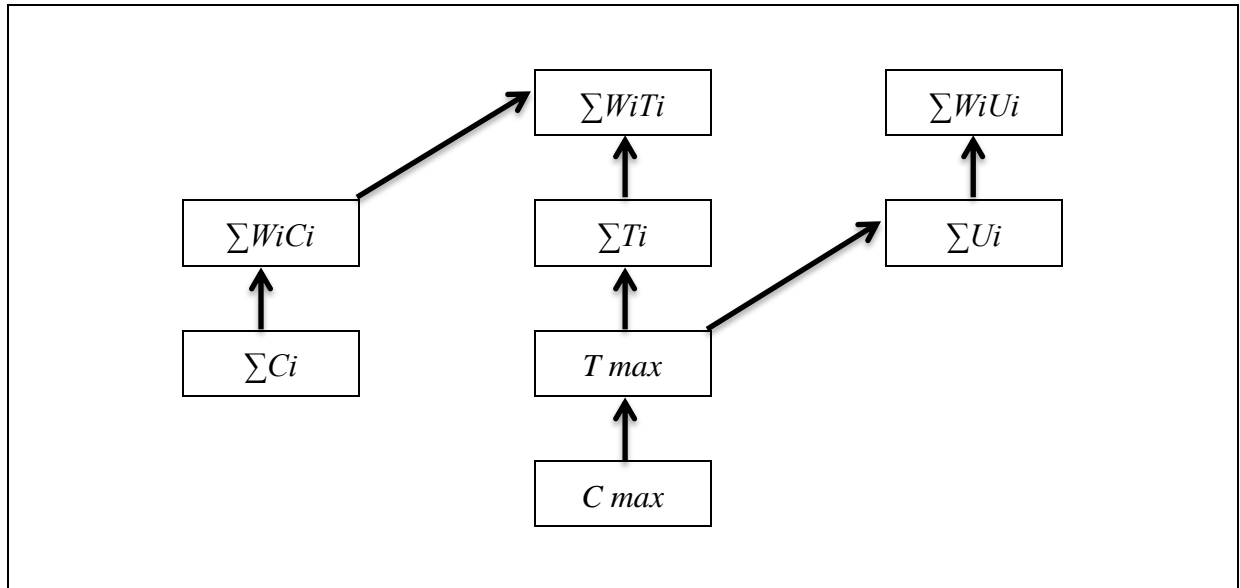


Figure 3 : Arbre de réduction

En effet, suivant sa complexité, le problème pourra ou non être résolu de façon optimale. Dans le cas de problèmes classés dans la classe **P** un algorithme polynomial a été mis en évidence, il suffit donc de l'utiliser, mais dans le cas de problèmes **NP**-difficile, deux méthodes sont offertes: les méthodes exactes et les méthodes approchées.

#### 4. Les Méthodes exactes

Elles sont issues de la recherche opérationnelle. Elles consistent à trouver la meilleure solution, et garantissent la complétude de résolution, basée sur le principe suivant.

##### 4.1. Principe :

La résolution se ramène à l'exploration de l'ensemble des choix d'affectation ou de séquence possibles au travers d'un arbre de recherche. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas). Pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Parmi ces techniques, on peut citer les différentes relaxations : la relaxation de base en programmation linéaire, la relaxation lagrangienne, la relaxation agrégée (surragote relaxation), et la décomposition lagrangienne. De plus, on emploie des heuristiques pour guider les choix de variables et de valeurs durant l'exploration de l'arborescence.

#### 4.2. Quelques méthodes exactes :

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années), en présentent quelques méthodes parmi les plus courantes et les plus connues :

##### 4.2.1. Enumération complète

L'énumération complète génère les ordonnancements un par un, en cherchant un ordonnancement optimal à travers tous les ordonnancements possibles. Dans le cas des problèmes d'ordonnancement non préemptifs sur une machine, il existe  $n!$  séquences d'exécution des tâches possibles. Par conséquent, pour le problème de machines correspondant, il y a  $(n!)^m$  séquences d'exécution possibles. Le nombre  $(n!)^m$  est très grand même pour des valeurs relativement petites de  $n$  et  $m$ . Et bien qu'en pratique, à cause des contraintes du problème, beaucoup d'ordonnements peuvent être inadmissibles et, que des procédures d'élimination peuvent parfois être utilisées pour voir si la non-optimalité d'un ordonnancement implique la non-optimalité d'autres non encore générés, le temps de résolution peut être prohibitivement long même si on utilise le plus puissant et le plus rapide des ordinateurs.

Il est donc clair que, chercher une solution optimale en utilisant l'énumération complète n'est pas convenable même pour des problèmes de petites tailles. C'est pour cela qu'il était nécessaire de faire appel à des méthodes intelligentes.

##### 4.2.2. Analyse combinatoire

L'analyse combinatoire peut mener à des algorithmes très efficaces qui produisent un ordonnancement optimal en un nombre prévisible d'étapes, où ce dernier grandit au plus polynomialement en fonction de la taille du problème.

Cette méthode examine l'effet d'un changement mineur sur une séquence particulière, sur la valeur de cette séquence.

Considérons par exemple l'argument de permutation de deux tâches adjacentes dans le problème  $(1 // \sum W_i C_i)$ .

Soit une séquence  $\sigma = \sigma_1 i j \sigma_2$ . En permutant dans  $\sigma$  les tâches  $i$  et  $j$ , nous obtenons ainsi la séquence  $\sigma' = \sigma_1 j i \sigma_2$ . Les dates de fin dans  $\sigma'$  sont les mêmes que celles de  $\sigma$  sauf pour les tâches  $i$  et  $j$ . Si  $H$  est la date de fin de 1, nous obtenons

$$\begin{aligned}
& \sum_{k=1}^n W_{\sigma'(k)} C_{\sigma'(k)} - \sum_{k=1}^n W_{\sigma(k)} C_{\sigma(k)} \\
&= W_j(h + p_j) + W_i(h + p_j + p_i) - W_i(h + p_i) + W_j(h + p_i + p_j) \\
&= (h + p_i + p_j)[W_i(1 - p_j) + W_j(1 - p_i)]
\end{aligned}$$

Par suite, nous ordonnons  $i$  avant  $j$ , si  $w_i p_j - w_j p_i \geq 0$  ou  $\frac{p_i}{w_i} \leq \frac{p_j}{w_j}$

#### 4.2.3. Programmation dynamique (P.D.)

La (P.D.) est fondée sur une approche récursive. Sommairement, elle s'applique à n'importe quel problème qui peut être divisé en une séquence de problèmes emboîtés.

Les problèmes résolus par la programmation dynamique se ramènent au choix d'une suite (finie ou infinie selon le problème) de décisions séquentielles, appelée politique, faisant évoluer l'état d'un système.

La (P.D.) (applicable sous certaines hypothèses) construit une politique optimale pour un critère [CAR 88]. Held et Karp [HEL62] suivirent les idées de Bellman et appliquèrent la (P.D.) aux problèmes d'ordonnements sur machine. Il a été ainsi prouvé que la (P.D.) est une bonne approche pour la résolution de certains problèmes d'ordonnements [ABD87].

Dans l'exemple ci-dessous, nous illustrons l'approche de la (P.D.) pour le cas ( $I / \sum f_j$ ), où on a à minimiser le coût total  $\sum_j f_j(C_j)$ .

Cette équation récursive a été suggérée par Held et Karp [HEL62].

Soient  $J \in I$ ,  $I = \{1, 2, \dots, n\}$  un ensemble arbitraire de tâches et  $f^*(J)$  le coût total minimal de l'ordonnement des tâches de  $J$  sur la période  $[0, \sum_{i \in J} p_i]$ .

L'objectif est de trouver  $f^*(I)$  en résolvant les équations récursives

$$f^*(j) = \min_{i \in j} \left\{ f^*(j - \{i\}) + f_i \left( \sum_{i \in j} p_i \right) \right\}$$

qui sont initialisées par  $f^*(\emptyset) = 0$ .

La (P.D.) est une méthode d'énumération implicite, meilleure que l'énumération complète, mais dont les besoins en espace mémoire sont souvent très grands.

**Remarque**

Les techniques de la méthode par séparation et évaluation ont été les approches non-heuristiques les plus couronnées de succès pour la résolution des problèmes d'ordonnancement, quoiqu'il existe des occasions où une résolution par la (P.D.) est à préférer (Baker et Schrage [BAK78]).

**4.2.4. Méthode par séparation et évaluation (Branch and Bound)**

Les techniques de cette méthode ont été développées et utilisées pour la première fois par Eastman [EAS59] pour le problème du voyageur de commerce, et par Land et Doig [LAN60] pour la programmation en nombres entiers.

Ces techniques ont été appliquées pour la première fois aux problèmes d'ordonnancement par Lomnicki [LOM65] et Ignall et Schrage [IGN65].

Les principes de base de cette méthode seront étudiés dans le cadre des problèmes d'ordonnancement de minimisation car nos sujets faits.

**L'arbre de recherche :** l'arbre de recherche est une représentation schématique permettant de recenser systématiquement et explicitement tous  $n!$  Permutations de  $n$  tâches. Il apparaît le voir comme un ensemble de nœuds disposés en couches hiérarchisées et numérotées respectivement de 0 à  $n-1$  du plus haut au plus bas niveau.

**Procédure de séparation :** cette procédure décrit la méthode utilisée pour partitionner un sous-ensemble de solution possible.

**Procédure de calcul d'une borne-inférieure :** cette procédure est indispensable dans une méthode par séparation et évaluation. Il s'agit en réalité d'une fonction définie sur l'ensemble des nœuds de l'arbre de recherche, et qu'on note ici par  $Lb$ . Si  $Y$  est un nœud quelconque, nous devrions être capable de  $Lb(Y)$  et avoir :  $Lb(Y) \leq f(Y) \quad \forall y \in Y$ .

**Borne supérieure :** si aucune solution admissible n'est connue, la borne supérieure  $UB$  est initialisée à l'infini jusqu'à ce que la première solution admissible soit trouvée.

**Stratégies de recherche :** par convention, le premier nœud à être exploré est le nœud-racine  $S$ . Une stratégie de recherche permet de sélectionner, dans l'arbre de recherche, le prochain nœud à explorer. Soit  $Y$  ce nœud recherché. Généralement, le nœud  $Y$  peut être choisi comme étant :

- le nœud ayant la plus petite borne-inférieure.
- le nœud le plus récemment créé.
- le nœud ayant la plus petite borne-inférieure parmi les nœuds les plus récemment créés.

La méthode par séparation et évaluation est un exemple typique de l'approche d'énumération implicite, qui peut trouver une solution optimale en examinant systématiquement des sous-ensembles de solutions admissibles.

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable, malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante. Dans la pratique, ce type de résolution n'est pas utilisable, hormis pour des cas très particuliers.

## 5. Les méthodes approchées

Les méthodes approchées issues de l'intelligence artificiel dont le but est de trouver une solution de bonne qualité, pour les problèmes d'optimisation de grande taille, en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue.

Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens. On peut citer les méthodes gloutonnes et l'amélioration itérative : par exemple, la méthode de Lin et Kernighan qui resta longtemps le champion des algorithmes pour le problème du voyageur de commerce.

Depuis une dizaine d'années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées métaheuristiques.

Une heuristique est une méthode, conçue pour un problème d'optimisation donné, qui produit une solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème.

Une métaheuristique est définie de manière similaire, mais à un niveau d'abstraction plus élevé. Le terme « métaheuristique » a été initialement utilisé par F.Glover pour distinguer la méthode tabou des heuristiques spécifiques. Notons que ce terme est également utilisé par J.L.Laurière dans son système de résolution Alice.

### 5.1. Les Méthodes heuristiques

Elles ont pour objet d'atténuer la combinatoire importante des méthodes exactes. Des solutions satisfaisantes sont ainsi obtenues en un temps raisonnable. On distingue notamment

- les méthodes adaptées des méthodes exactes qui s'inspirent des techniques d'optimisation précédentes en restreignant leur champ d'application. La décomposition du



problème global en une succession de sous problèmes de taille réduite, où une résolution exacte peut être effectuée, est souvent employée.

- les méthodes de placement consistent à sélectionner les ordres de fabrication successivement suivant une règle de priorité. Les opérations d'un ordre de fabrication sont ensuite positionnées dans le temps en tenant, compte des opérations déjà placées. De nombreux progiciels (SAVEPLAN par exemple) utilisent ce type de méthode.

- les méthodes sérielles dites aussi de simulation, affectent les tâches aux machines en fonction des dates de disponibilité des machines et selon des règles de priorité. Des logiciels à événements discrets permettent de simuler l'avancement des opérations en gérant les conflits entre opérations intervenant dans les files d'attente devant chaque machine par des règles de priorité.

Quelques progiciels (TZARII, PARSIFAL, SIPA PLUS par exemple) utilisent les méthodesérielles.

Les méthodes heuristiques ont l'avantage de fournir une solution très rapidement. Elles sont donc bien adaptées pour être utilisées dans une boucle de décision avec l'utilisateur. Leur efficacité est liée alors à la capacité de l'utilisateur à bien orienter la résolution.

### **5.2. Les Méta-heuristiques**

Elles consistent à explorer « intelligemment » l'espace des solutions possibles pour trouver l'optimum ou une solution la plus proche possible.

Elles ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire. Les métaheuristiques sont représentées essentiellement par : les méthodes de **voisinage**, par exemple on a:

#### **5.2.1. La recherche avec tabous**

L'idée de base consiste le meilleur voisin peut être accepté même si sa valeur est plus petite que le point d'origine. Cependant, cette stratégie peut entraîner des cycles. Afin de les éviter, on mémorise une liste, dite taboue, des dernières variables ayant été modifiées par les étapes précédentes. Cette liste est ensuite utilisée pour interdire de modifier les variables.

#### **5.2.2. Le recuit simulé (simulated annealing)**

Son fonctionnement est inspiré du phénomène physique de cristallisation. Lorsqu'un métal en fusion se refroidit, les molécules qui le composent vont petit à petit se structurer selon une conformation de basse énergie. Le recuit simulé commence par une exploration par un cheminement proche de l'aléatoire (température haute), puis au cours des itérations, sa

fonction de déplacement est graduellement modifié pour tendre de plus en plus vers des déplacements ascendants, en analogie avec le métal qui se refroidi petit a petit. Ce principe est généralement présent du point de vue de la minimisation d'une fonction (énergie basse), mais dans la continuité de notre présentation, nous le décrivons comme un algorithme de maximisation.

Les algorithmes **évolutifs**, parmi les plus courantes et les plus connues :

### 5.2.3. Les algorithmes génétiques

Permet d'explorer une plus grande partie de région de solution en combinant des solutions réalisables. Un ordonnancement obtenu par une simulation à événements discrets est amélioré au moyen d'un algorithme génétique. Une approche par algorithmes génétiques est aussi utilisée pour améliorer un ordonnancement multi-objectif.

Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle.

En conclure une typologie générale de méthodes de résolutions dans la figure suivante :

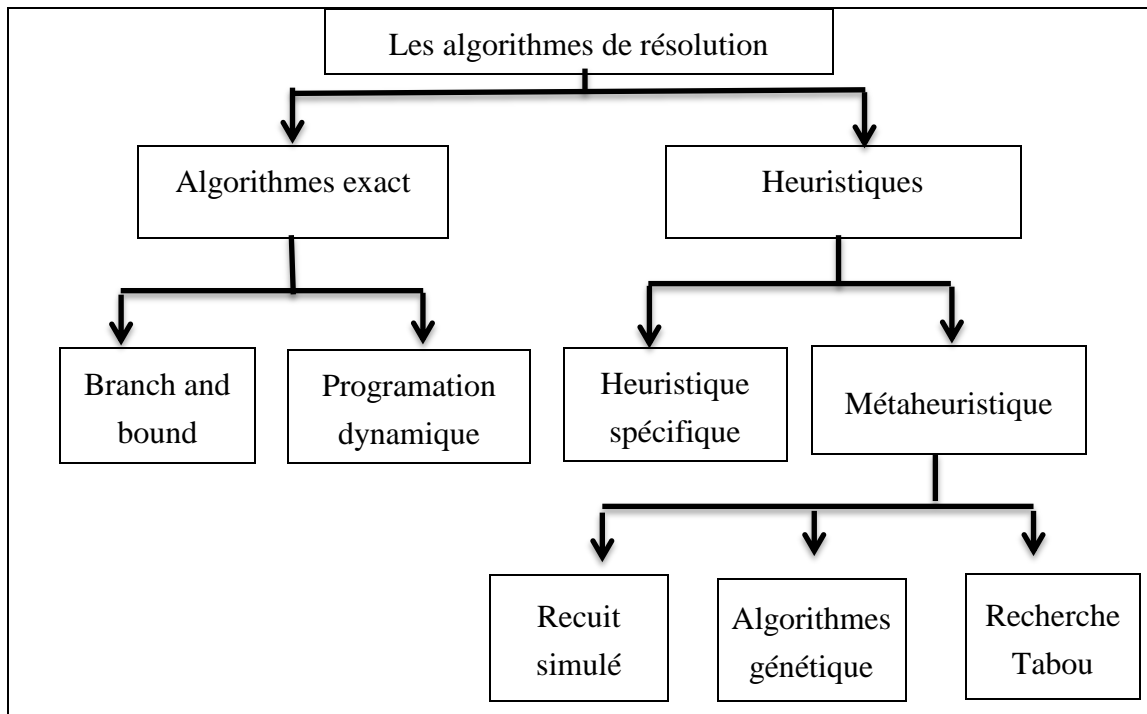


Figure 4 : Classification des algorithmes de résolution

## 6. Méthode exacte contre méthode approchée

En peut faire une petite comparaison entre les méthodes de résolution exacts et approchées, les résultats sont détaillée sur le tableau :

Méthode exacte	Méthode approchée
Efficacité pour des cas très particuliers des problèmes taille raisonnable	Efficacité pour des problèmes de plus grande taille
Nécessite d'espace mémoire non raisonnable	Nécessite d'espace mémoire relativement raisonnable par rapport aux méthodes exactes
La résolution nécessite une durée de temps non raisonnable	La résolution nécessite une durée de temp très petit par rapport aux méthodes exactes
Garantie d'optimalité	Aucune garantie d'optimalité
Pratiquement Difficile à implanter et à la compréhension	Pratiquement simple à implanter et à la compréhension

Alors les méthodes approchées généralement sont les plus utilisée mais on a :

### *Avantages d'une méthode approchée*

- ✓ Flexible aux variations du problème
- ✓ Changement de données, contraintes, objectifs
- ✓ Robuste
- ✓ Facile à comprendre et à implémenter
- ✓ Solution de bonne qualité rapidement

### *Inconvénient d'une méthode approchée*

- X Aucune garantie d'optimalité
- X Difficile d'analyser les performances
- X Phase de calibrage des paramètres
- X Concevoir une méthode efficace : de l'art et du savoir-faire !