REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi

**Cours Méthodes algorithmiques**

**1ere année master AD**

# Chapitre 6 : Les méthodes approximatives

# Introduction aux Méthodes Approximatives

- Les **méthodes approximatives** désignent une catégorie d'approches algorithmiques qui cherchent à trouver des solutions proches de la solution optimale à un problème, mais sans nécessairement garantir l'optimalité exacte.

- Ces méthodes sont particulièrement utiles pour résoudre des problèmes complexes où les méthodes exactes sont inefficaces ou impraticables en raison de leur coût en temps ou en espace, surtout dans les cas de grands ensembles de données ou de problèmes NP-difficiles.

# Définition des Méthodes Approximatives

- Les algorithmes approximatifs visent à produire une solution dont la qualité est garantie par un facteur d'approximation.
- Il existe une borne supérieure ou inférieure connue qui quantifie à quel point la solution trouvée s'éloigne de l'optimum.

# Rôle des Méthodes Approximatives



# Rôle des Méthodes Approximatives

Pour les problèmes NP-difficiles, les algorithmes exacts peuvent ne pas exister, ou bien être très coûteux en termes de temps d'exécution. Les méthodes approximatives offrent alors une alternative permettant de trouver des solutions acceptables en un temps raisonnable.

NP (Non-deterministic Polynomial time) désigne l'ensemble des problèmes pour lesquels une solution donnée peut être **vérifiée en temps polynomial** par un algorithme déterministe



# Rôle des Méthodes Approximatives

Ces méthodes sont souvent plus simples à mettre en œuvre et peuvent être exécutées plus rapidement que les méthodes exactes, ce qui les rend adaptées à des applications en temps réel ou à des systèmes ayant des ressources limitées



# Rôle des Méthodes Approximatives

De nombreuses méthodes approximatives sont accompagnées de garanties sur l'écart entre la solution trouvée et la solution optimale. Ces garanties permettent de quantifier la qualité des solutions proposées.



# Concepts Clés des Méthodes Approximatives en Algorithmique

Les  méthodes approximatives s'appuient sur des notions théoriques précises afin d'évaluer la qualité et l'efficacité des solutions proposées. Voici les principaux concepts:



# Concepts Clés des Méthodes Approximatives en Algorithmique

## 1. Ratio d'approximation

Le ratio d'approximation est un indicateur central permettant de quantifier la performance d'un algorithme d'approximation. Il évalue la qualité d'une solution obtenue par rapport à une solution optimale. Supposons que :

$C_{approx}$ désigne le coût (ou la valeur) de la solution produite par l'algorithme d'approximation

$C_{opt}$ représente le coût (ou la valeur) de la solution optimale.

**Le ratio d'approximation (RA) est défini différemment selon la nature du problème :**

- **Minimisation**
- **Maximisation**

# Concepts Clés des Méthodes Approximatives en Algorithmique

## 1. Ratio d'approximation

Pour un problème de **minimisation** :

$$RA = \frac{C_{approx}}{C_{opt}}, \quad \text{avec } RA \geq 1.$$

Pour un problème de **maximisation** :

$$RA = \frac{C_{opt}}{C_{approx}}, \quad \text{avec } RA \leq 1.$$

# Concepts Clés des Méthodes Approximatives en Algorithmique

## 1. Ratio d'approximation

Un algorithme est dit α-approximatif si RA≤ α pour tous les cas d'entrée, où α>1 est une constante.

**Exemple explicatif :**

Dans un problème de sac à dos fractionnaire, si l'algorithme donne une solution avec :

$C_{approx}=120$

Et la solution optimale est $C_{opt}$ =100, alors RA=100／120=0.83

Donc, la solution obtenue par l'algorithme est à 83.3% de l'optimum, ce qui est une mesure plus logique pour un problème de maximisation.

# Concepts Clés des Méthodes Approximatives en Algorithmique

## 2. Garanties de performance

Un algorithme d'approximation ne vise pas nécessairement à atteindre une solution optimale, mais il garantit que la solution trouvée est proche de l'optimum dans un intervalle contrôlé. Les garanties de performance se définissent comme suit :



# Concepts Clés des Méthodes Approximatives en Algorithmique

## 2. Garanties de performance

**Limite supérieure**

Dans les problèmes de minimisation, l'algorithme garantit que :

$$C_{approx} \leq \alpha \cdot C_{opt}$$

**Limite inférieure**

Pour les problèmes de maximation , l'algorithme garantit que :

$$C_{approx} \geq C_{opt} / \alpha$$

Ces garanties permettent d'évaluer la robustesse de l'algorithme indépendamment de la complexité spécifique de l'entrée.

# Classification des algorithmes d' Approximation

La classification des algorithmes d’approximation selon leur **facteur d’approximation** signifie comment ces algorithmes approchent de la solution optimale (OPT) d’un problème difficile à résoudre exactement

# Classification des algorithmes d' Approximation

## 1. PTAS (Polynomial Time Approximation Scheme)

Dans un **PTAS**, l'utilisateur peut choisir un facteur d'approximation $(1+\epsilon)$, où $\epsilon>0$.

Plus $\epsilon$ est petit (plus la solution est proche de l'optimum), plus le temps d'exécution de l'algorithme augmente.

Le problème du sac à dos peut être approximé par un PTAS

# Classification des algorithmes d' Approximation

## 1. PTAS (Polynomial Time Approximation Scheme)

Dans un **PTAS:**

Le temps d'exécution polynomial en fonction de la taille de l'instance du problème

La Précision de l'approximation, l'approche trouve une solution qui est proche de la solution optimale, avec une garantie de performance liée à un paramètre d'approximation $\epsilon$ , où $0<\epsilon<1$.

Le problème du sac à dos peut être approximé par un PTAS

# Classification des algorithmes d' Approximation

## 1. PTAS (Polynomial Time Approximation Scheme)

Tu veux une solution **très proche de l'optimum:**

- Si tu acceptes **5% d'erreur** (ε = 0.05), l'algorithme sera un peu lent
- Si tu veux **1% d'erreur** (ε = 0.01), il sera plus lent
- Si tu veux **0.1%**, il deviendra encore plus lent



- **Plus la précision est forte, plus l'algorithme ralentit**
- **Mais pour chaque ε fixé, le temps reste polynomial**

# Classification des algorithmes d' Approximation

## 2. FPTAS (Fully Polynomial Time Approximation Scheme)

Le **FPTAS** est une version plus rapide du PTAS.

Ici, le temps d'exécution est polynomial en fonction de la taille de l'entrée ET de $1 \setminus \epsilon$ .

Exemple : Le problème du sac à dos fractionnaire.

# Classification des algorithmes d' Approximation

## 2. FPTAS (Fully Polynomial Time Approximation Scheme)

Pour Le **FPTAS**

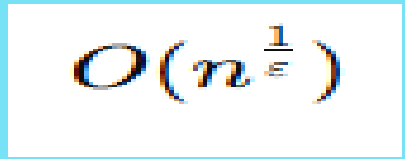
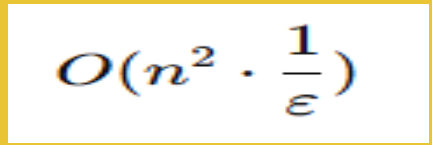
Temps d'exécution polynomial par rapport à la taille de l'entrée (en n) et à $1/\epsilon$

Précision de l'approximation : L'algorithme doit trouver une solution dont la valeur est au moins $(1-\epsilon)$fois la valeur optimale, où $\epsilon$ est un paramètre d'approximation. Cela signifie que la solution obtenue par l'algorithme est proche de l'optimum.

Exemple : Le problème du sac à dos fractionnaire.

# Classification des algorithmes d' Approximation

## PTAS VS FPTAS

| TYPE | Temps d'exécution |
|---|---|
| PTAS | Polynomial quand ε est fixe, mais *peut devenir exponentiel en 1/ε*  |
| FTPAS | Polynomial en n ET en 1/ε (donc toujours polynomial)  |

# Classification des algorithmes d' Approximation

## FPTAS VS PTAS

Supposons n = 100 objets et souhaitons un résultat avec :

- **ε = 0.1 (10 % d'erreur maximum)**

$$\text{PTAS} \rightarrow \text{temps} : n^{1/\varepsilon} = 100^{10} = \text{impossible}$$

$$\text{FPTAS} \rightarrow \text{temps} : n^3/\varepsilon = 100^3/0.1 = 10^7 \rightarrow \text{faisable}$$

- **ε = 0.01 (1 % d'erreur)**

$$\text{PTAS} \rightarrow \text{temps} : n^{1/\varepsilon} = 100^{100} = \text{gigantesque}$$

$$\text{FPTAS} \rightarrow \text{temps} : n^3/\varepsilon = 10^8 \rightarrow \text{toujours faisable}$$

# Classification des algorithmes d' Approximation

**FPTAS VS PTAS**



# Classification des algorithmes d' Approximation

**Exemple concret :**

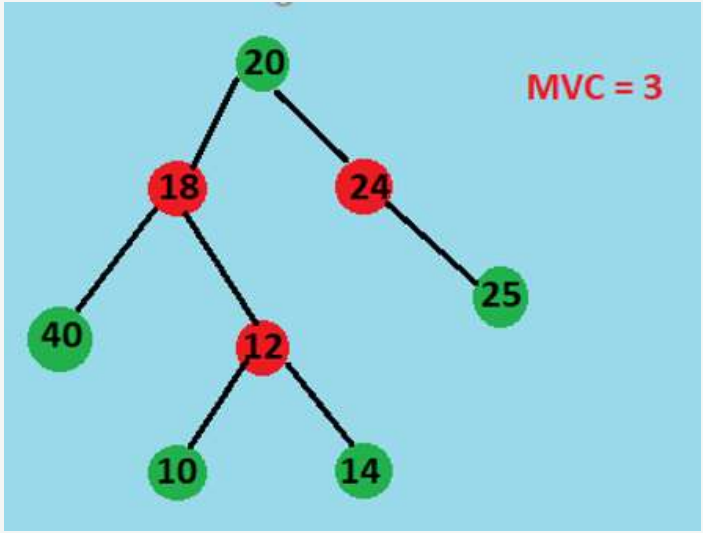
**Le problème du sac à dos**

- **Exact :** Résolution complète en temps pseudo-polynomial avec programmation dynamique.
- **PTAS :** Résolution approximative en temps polynomial rapide pour $\epsilon>0$.
- **FPTAS :** Approche similaire, mais le temps peut augmenter plus vite avec $\epsilon$.

# exemples classiques d'algorithmes d'approximation

## Le Problème de la Couverture de Sommets (Vertex Cover)

consiste à déterminer le plus petit ensemble de sommets dans un graphe G=(V,E) tel que chaque arête de E soit incidente à au moins un sommet du sous-ensemble de couverture. Ce problème est également NP-difficile.



# exemples classiques d'algorithmes d'approximation

## Le Problème de la Couverture de Sommets (Vertex Cover)

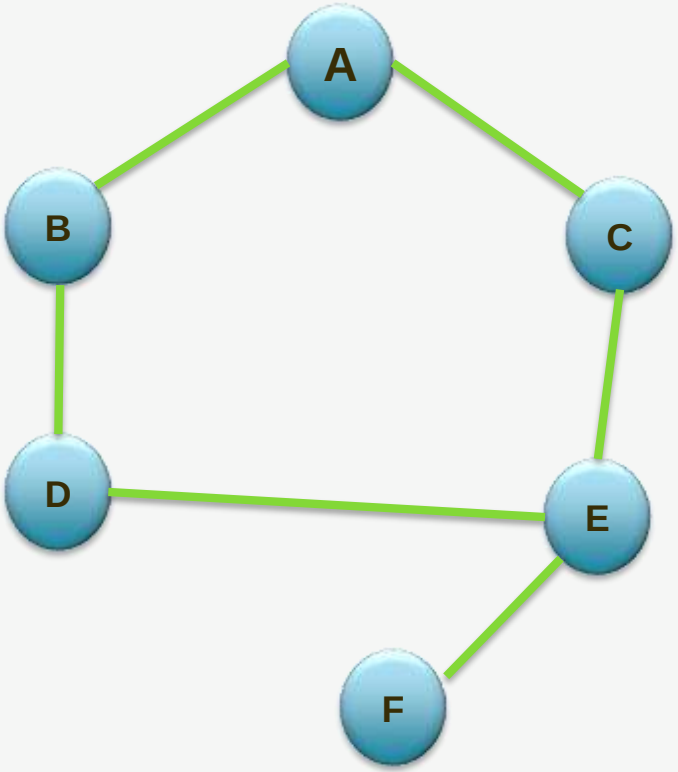
**Algorithme approximatif : Algorithme glouton**

Une approche d'approximation consiste à utiliser un algorithme glouton, qui fonctionne de la manière suivante :

1. Choisir une arête non couverte arbitraire dans le graphe.
2. Ajouter les deux sommets de l’arête au sous-ensemble de couverture.
3. Supprimer toutes les arêtes couvertes par ces deux sommets.
4. Répéter jusqu'à ce que toutes les arêtes soient couvertes.

# exemples classiques d'algorithmes d'approximation

## Le Problème de la Couverture de Sommets (Vertex Cover)



## Le Problème de la Couverture de Sommets (Vertex Cover)

Nous avons couvert (A,B), (A,C)et (B,D) avec les sommets A et B.
Nous avons couvert (C,E), (D,E), et (E,F) avec les sommets C et E.

Le sous-ensemble de couverture final est **{A,B,C,E},** et toutes les arêtes sont couvertes.

## Le Problème de la Couverture de Sommets (Vertex Cover)

**Analyse du ratio d'approximation :**

- Cet algorithme fournit une solution qui est au plus deux fois plus grande que la solution optimale,
- c'est-à-dire qu'il est 2-approximatif.
- Pour un ensemble de couverture optimal $C_{opt}$, la taille de la couverture produite par cet algorithme est $2 \times |Copt|$

## Le Problème de la Couverture de Sommets (Vertex Cover)

**Solution optimale :**

Le graphe comporte 6 arêtes, et chaque arête doit être couverte par au moins un sommet de l'ensemble de couverture.

L'ensemble optimal de sommets pourrait être constitué de **B**, **E** (par exemple), car ces sommets couvrent toutes les arêtes.

**B** couvre (A,B), (B,D)
**E** couvre (C,E), (D,E), (E,F)

# exemples classiques d'algorithmes d'approximation

## Le Problème de la Couverture de Sommets (Vertex Cover)

```
def couverture_gloutonne(G):
        Sommets = set()
        ARRETE_couv = set()

    while len(ARRETE_couv) < len(G.edges):

        (u, v) = choisir_une_arrete_non_couverte(G, ARRETE_couv)

        Sommets.add(u)
        Sommets.add(v)


        for (x, y) in G.edges:
            if x == u or y == u or x == v or y == v:
                ARRETE_couv.add((x, y))

    return Sommets
```

# exemples classiques d'algorithmes d'approximation

## Le Problème du Sac à Dos Fractionnaire (Fractional Knapsack Problem)

- un problème d'optimisation dans lequel on a un sac à dos avec une capacité maximale et un ensemble d'objets, chacun ayant un poids et une valeur.

- L'objectif est de maximiser la valeur totale dans le sac à dos, tout en respectant la contrainte de capacité. Contrairement au sac à dos classique, où chaque objet doit être pris en entier, dans le sac à dos fractionnaire, les objets peuvent être fractionnés.

## Le Problème du Sac à Dos Fractionnaire (Fractional Knapsack Problem)

**Algorithme approximatif : Algorithme glouton**

L'algorithme glouton pour ce problème :

1. Calculer le rapport valeur/poids pour chaque objet.
2. Trier les objets en fonction de ce rapport, du plus élevé au plus faible.
3. Ajouter des objets au sac à dos, en commençant par celui avec le meilleur rapport valeur/poids.
4. Si un objet ne peut pas être ajouté dans sa totalité, ajouter la fraction de l’objet qui reste dans le sac à dos jusqu'à ce qu’il soit plein.

## Le Problème du Sac à Dos Fractionnaire (Fractional Knapsack Problem)

**Analyse du ratio d'approximation :**

- L'algorithme fournit une solution optimale pour le problème du sac à dos fractionnaire.
- Le ratio d'approximation est **1** pour ce problème particulier.

Par exemple si un objet a une valeur de 60 et un poids de 10, et que la capacité du sac à dos est de 50, l'algorithme pourrait prendre 5 objets complets et la fraction de l'objet suivant, produisant une solution optimale avec une valeur totale maximale.

# Avantages et limites des méthodes approximatives

## Les avantages

| | |
|---|---|
| **Réduction de la Complexité Computationnelle** | Permet de résoudre des problèmes NP-difficiles en un temps raisonnable, là où les algorithmes exacts sont impraticables. |
| **Adaptabilité aux Problèmes NP-Difficiles** | Efficace pour les problèmes où une solution exacte est difficile à obtenir comme le problème du sac à dos et de couverture de sommets |
| **Simplicité d'Implémentation** | Simple à implémenter, notamment pour les algorithmes gloutons, sans nécessiter de structures complexes. |
| **Solutions Pratiques Suffisantes** | Fournit souvent une solution suffisamment bonne dans des applications pratiques où l'optimalité n'est pas strictement nécessaire. |

# Avantages et limites des méthodes approximatives

## Les limites

| | |
|---|---|
| **Absence de Garantie d'Optimalité** | Ne garantit pas une solution optimale : un algorithme glouton peut donner une solution bien plus coûteuse |
| **Performance Variable** | L'efficacité varie selon la structure du problème et peut ne pas être aussi performante dans tous les cas. |
| **Ratio d'Approximation Limité** | Les solutions peuvent être éloignées de l'optimal, surtout avec des bornes élevées sur le ratio d'approximation par ex problème du voyageur de commerce |