SIO - Simulation et optimisation

Heuristiques et approximation

13 octobre 2025

Table des matières

1 Heuristiques

1.1 Optimisation combinatoire

L'optimisation dite combinatoire est caractérisée par un ensemble de solutions finies mais souvent très grand. Nous aurons donc les caractéristiques suivantes:

- un ensemble **fini** S de solutions admissibles ou réalisables
- une fonction objectif $f : S \to \mathbb{R}$ associant une valeur à chaque solution

> **i Info**
>
> On cherchera donc souvent à résoudre le problème d'optimisation suivant:
>
> $$\min_{x \in S} f(x)$$

1.1.1 Type de problèmes

- problèmes `faciles` (polynomiaux): on peut trouver une solution optimale en temps polynomial
- problèmes `difficiles` (NP-difficiles): on ne connaît pas d'algorithme polynomial pour trouver une solution optimale

1.1.2 Exemples

- **problème de l'arbre recouvrant de poids minimal** → problème polynomial
- **problème du plus court chemin de s à t** → problème polynomial grâce à l'algorithme de Dijkstra ou Bellman-Ford
- **problème du voyageur de commerce (TSP)** → problème NP-difficile
- **problème du stable de cardinal maximum** → problème NP-difficile
- **problème d'optimisation linéaire**
  - si les variables sont continues → problème polynomial
  - si les variables sont entières → problème NP-difficile

1.2 Algorithmes de résolution

1.2.1 Algorithmes exacts

- un algortihme `exact` fournit toujours une solution optimale
- pour les problèmes faciles, un bon algorithme `exact` a une complexité polynomiale et demande donc un temps de résolution borné par un polynôme en la taille du problème
- pour les problèmes difficiles, un algorithme `exact` a souvent une complexité exponentielle et demande donc un temps de résolution très long

1.2.2 Algorithmes approximatifs

- un algorithme `approximatif` fournit une solution réalisable mais pas forcément optimale
- elle assure la qualité minimale de la solution produite
- elle est de complexité raisonnable (polynomiale)

1.2.3 Algorithmes heuristiques

- une heuristique fournit une solution sous-optimale en général
- aucune garantie sur la qualité de la solution
- elle est efficace en pratique (temps de calcul raisonnable)
- on constate empiriquement qu'elle fournit souvent de bonnes solutions

> **💡 Hint**
>
> Une bonne heuristique doit posséder les qualités suivantes:
>
> - complexité raisonnable et relativement simple à implémenter
> - produire de bonnes solutions en pratique proche de l'optimal tout en minimisant les cas de mauvaises solutions

1.2.3.1 Heuristiques constructives

Les heuristiques constructives construisent une solution réalisable pas à pas en ajoutant des éléments à une solution partielle jusqu'à obtenir une solution complète. Ces méthodes ne disposent pas d'une solution complète du problème mais parfois d'une solution partielle.

1.2.3.2 Heuristiques d'amélioration

Les heuristiques d'amélioration ou dite d'échange, partent d'une solution admissible et cherchent à l'améliorer en explorant son voisinage. Ces modifications, souvent appelés échanges, sont enchaînées tant que des améliorations sont possibles.

1.3 Problème du bin packing

Le problème du bin packing (ou problème de l'emballage dans des conteneurs) consiste à regrouper un ensemble d'objets de tailles différentes dans un nombre minimal de conteneurs (ou « bins ») de capacité fixe, de manière à ce que la somme des tailles des objets dans chaque conteneur ne dépasse pas la capacité de celui-ci.

2 Heuristiques de coloration

2.1 Coloration d'un graphe

Soit $G = (V, E)$ un graphe non orienté. Une **coloration** de G est une affectation de couleurs aux sommets de G telle que deux sommets adjacents n'ont pas la même couleur. Le but est de minimiser le nombre de couleurs utilisées.

2.2 Heuristique LF (largest-first)

Cette heuristique consiste à ordonner les sommets par ordre décroissant de degré et à les colorier dans cet ordre.

Il s'agit d'une des approches algorithmiques les plus simples.

> ⚠ **Warning**
>
> **Comme toutes les colorations gloutonnes**, cette heuristique utilise au plus $\Delta(G) + 1$ couleurs pour un graphe G dont le degré maximal des sommets est $\Delta(G)$.

2.2.1 Exemple

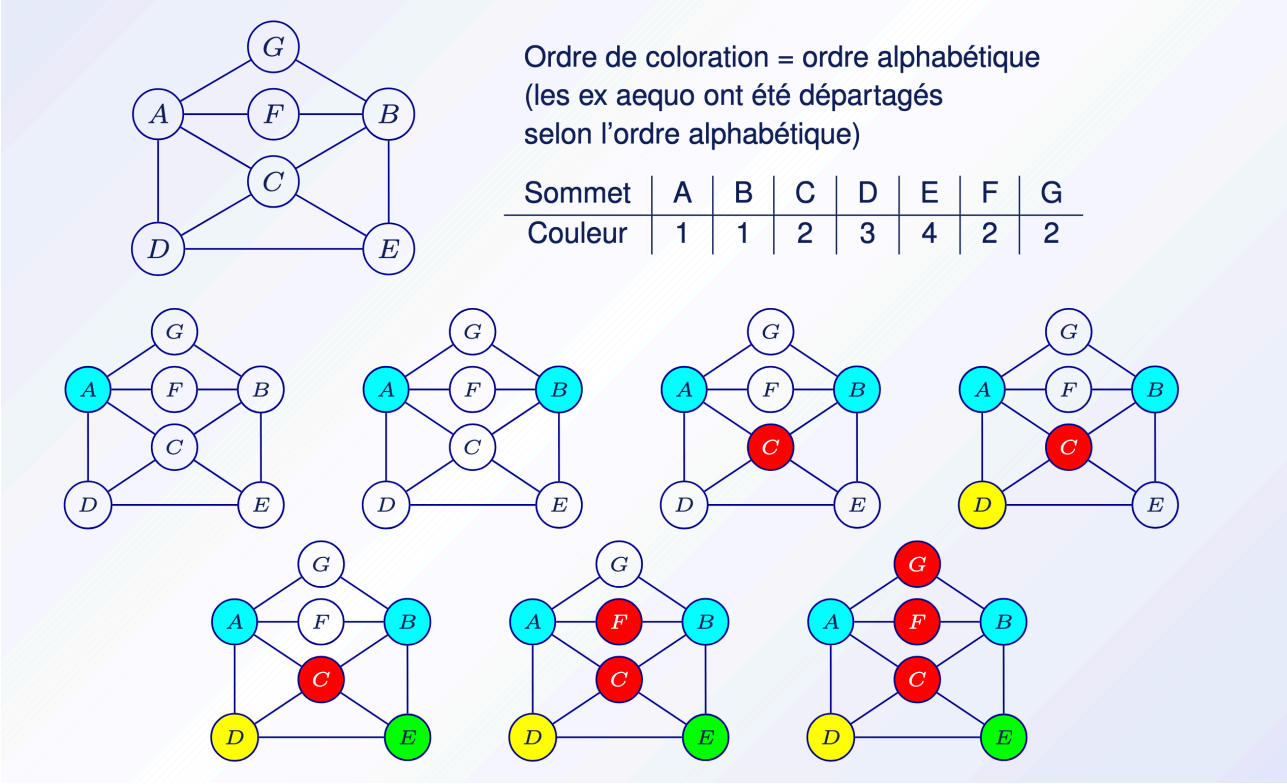

Fig. 1. – Capture des slides du cours – Exemple de la heuristique LF

2.3 Heuristique SL (smallest-last)

Cette heuristique consiste à colorier le plus petit sommet (de degré minimal) dans le sous-graphe restant, puis à le retirer du graphe. On répète ce processus jusqu'à ce que tous les sommets soient colorés.

La coloration séquentielle du graphe s'effectue ensuite dans l'ordre inverse de suppression des sommets.

2.3.1 Exemple

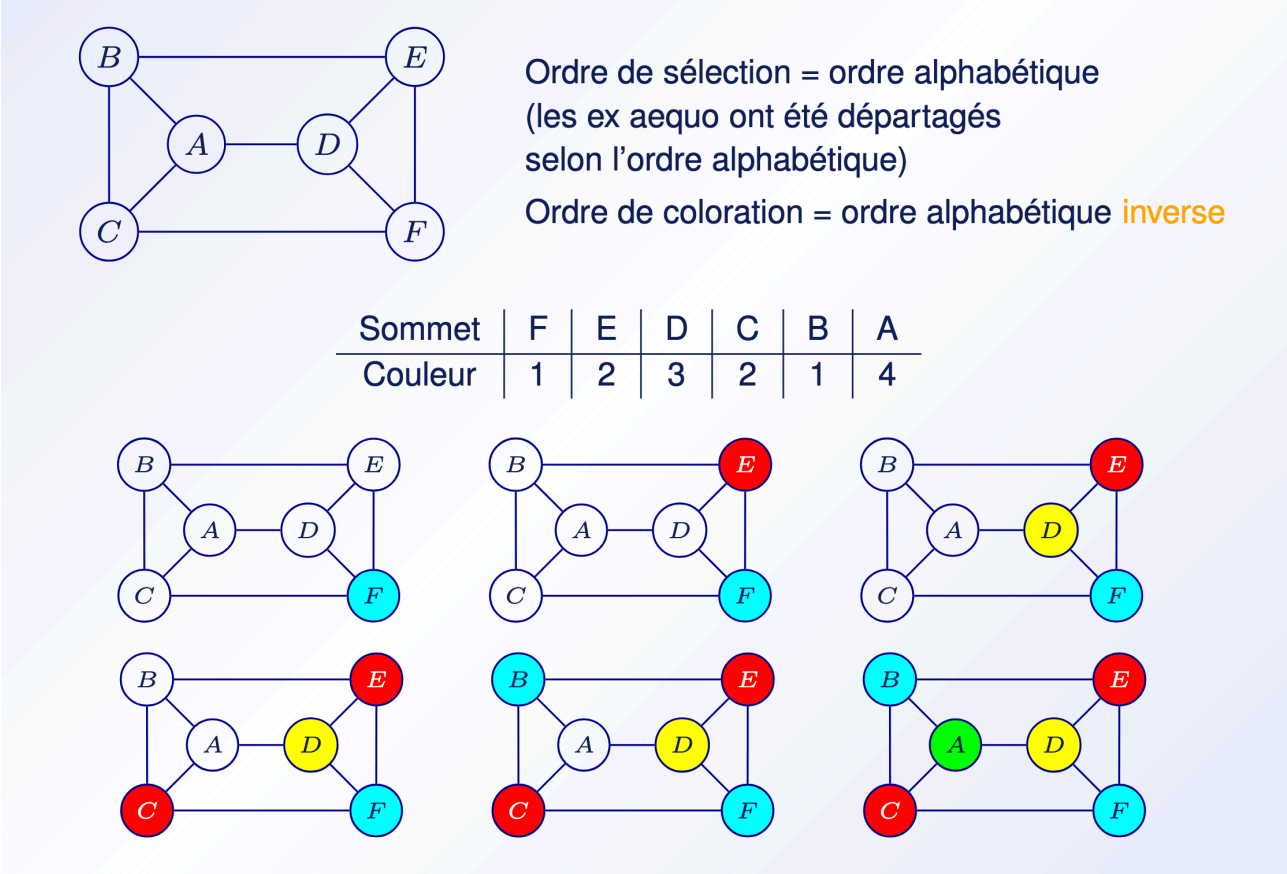

Fig. 2. – Capture des slides du cours – Exemple de la heuristique SL

2.4 Heuristiques DSATUR

L'heuristique DSATUR (Degree of Saturation) consiste à colorier les sommets en fonction de leur degré de saturation, c'est-à-dire le nombre de couleurs différentes déjà utilisées par leurs voisins.

> **i Info**
>
> L'algorithme commence par affecter une couleur au sommet de degré maximal. Ensuite, à chaque étape, il choisit le sommet non colorié avec le plus grand degré de saturation et lui attribue la couleur la plus basse possible.

> **Quote**
>
> Le degré de saturation d'un sommet est le nombre de **couleurs différentes** utilisées par ses **voisins déjà coloriés**.

2.4.1 Exemple

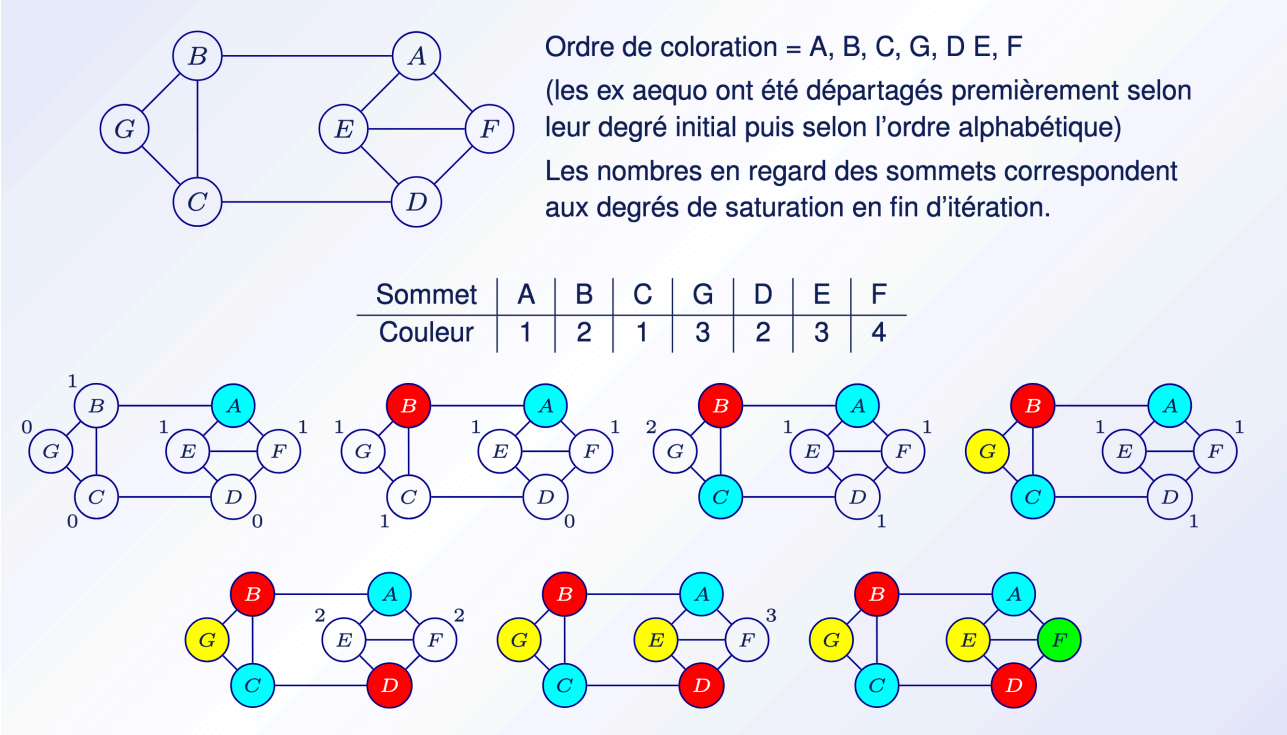

Fig. 3. – Capture des slides du cours – Exemple de la heuristique DSATUR

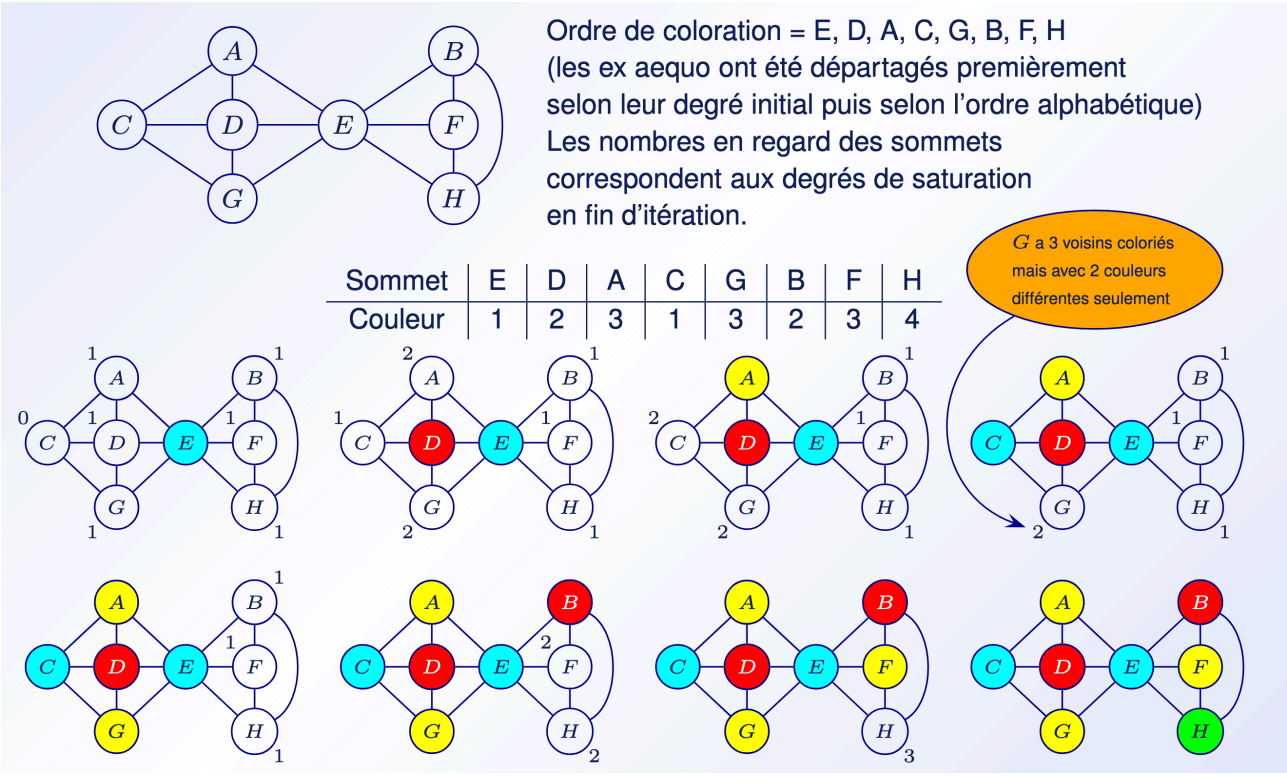

Fig. 4. – Capture des slides du cours – Exemple 2 de la heuristique DSATUR

2.5 Heuristiques RLF (recursive largest-first)

L'heuristique RLF (Recursive Largest First) est une méthode de coloration de graphe qui construit séquentiellement une partition du graphe en sous-ensembles stables $\{C_1, ..., C_k\}$, chaque ensemble C_i stable recevant la couleur i

2.5.1 Pseudo-code

```
Debut
(1) Déterminer le sommet v ∈ V de degré maximal dans G
(2) Poser C := {v}, U := Adj[v] et
      W := V \ (Adj[v] ∪ {v}) // Retirer v et ses voisins de l'ensemble V
(3) Tant que W n'est pas vide faire
(4)   Déterminer le sommet w ∈ W ayant un nombre maximal de voisins dans U
        // w est le sommet ayant le plus de voisins communs
        // avec les sommets déjà dans C ⟹ on l'ajoute à C
(5)   Poser C := C ∪ {w}, U := U ∪ Adj[w] et
        W := W \ (Adj[w] ∪ {w}) // Retirer w et ses voisins de W
(6) Retourner C
Fin
```

2.6 Remarques sur les performances

1. Les quatre heuristiques de coloration séquentielle présentées utilisent au plus $\Delta(G) + 1$ couleurs pour un graphe G dont le degré maximal des sommets est $\Delta(G)$.
2. Cette performance n'est pas spécifique aux méthodes présentées mais est vérifiée par tous les algorithmes de coloration gloutonne.
3. Les heuristiques DSATUR et RLF sont optimales pour les graphes bipartis (elles n'utilisent jamais plus de deux couleurs pour colorier de tels graphes).
4. Les performances empiriques des quatre méthodes correspondent à leur ordre de présentation, les deux dernières étant souvent clairement supérieures aux deux premières en pratique mais également plus lentes à s'exécuter et plus gourmandes en espace mémoire.

3 Heuristiques pour le TSP

Lors-ce que l'on parle de heuristiques pour le TSP on parle du problème du voyageur de commerce (Traveling Salesman Problem).

Objectif: Trouver la tournée la plus courte visitant chacune des villes une et une seule fois.

Mathématiquement, le problpme du voyageur de commerce consiste à trouver une permutation σ des n villes qui minimise la longueur totale de la tournée.

> **i Info**
>
> Le problème du voyageur de commerce est NP-difficile et à l'heure actuelle, on ne connaît pas d'algorithme de complexité polynomiale (dans tous les cas) permettant de trouver la meilleure tournée.

3.1 Plus proche voisin

L'heuristique du plus proche voisin est une méthode simple pour construire une solution approximative au problème du voyageur de commerce. Elle consiste à partir d'une ville initiale et à visiter à chaque étape la ville la plus proche non encore visitée jusqu'à ce que toutes les villes aient été visitées.

3.1.1 Algorithme

```
1. Choisir une ville de départ aléatoirement.
2. Choisir pour la prochaine destination la ville la plus proche non encore visitée.
3. Répéter l'étape 2 jusqu'à ce que toutes les villes aient été visitées.
4. Retourner à la ville de départ pour compléter la tournée.
```

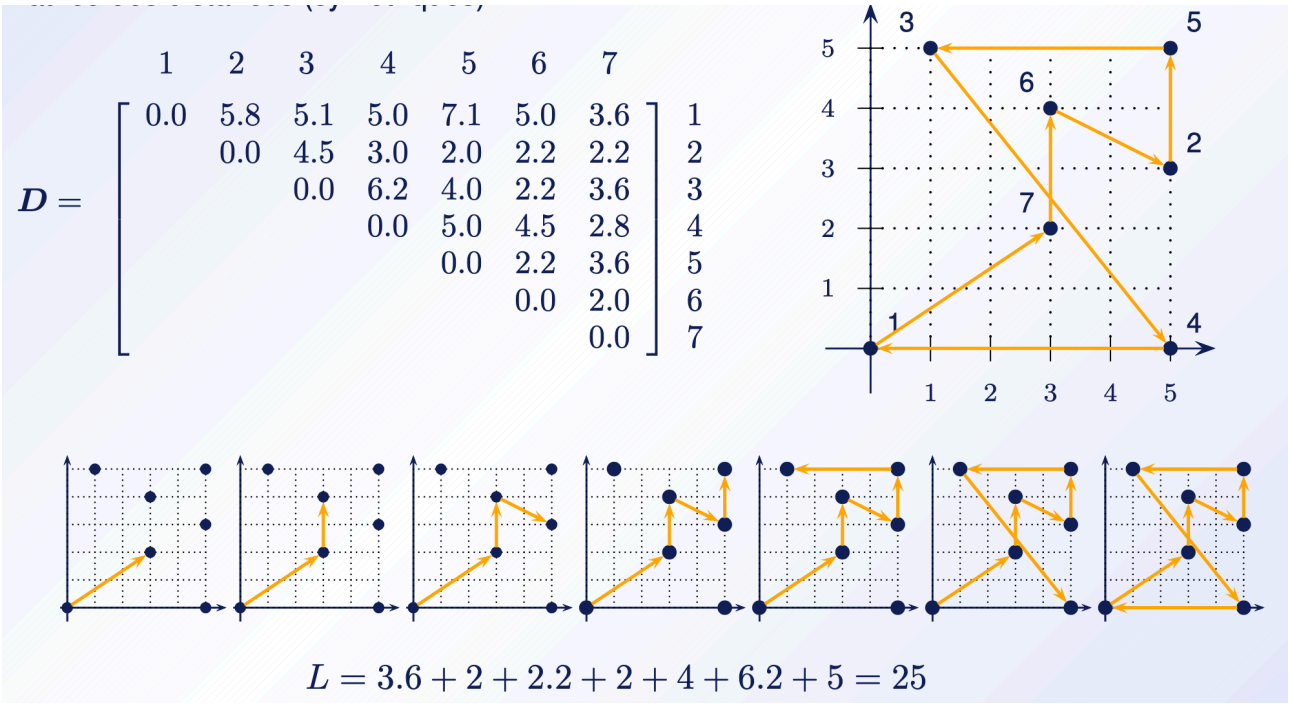

Fig. 5. – Capture des slides du cours – Exemple de l'heuristique du plus proche voisin

3.2 Heuristique gloutonne

L'heuristique gloutonne pour le TSP consiste à construire une tournée en ajoutant à chaque étape l'arête de coût minimal qui ne crée pas de cycle prématuré et qui n'augmente pas le degré des sommets au-delà de 2, jusqu'à ce que toutes les villes soient connectées.

3.2.1 Algorithme

```
1. Initialiser un ensemble vide d'arêtes pour la tournée.
2. Trier toutes les arêtes du graphe par ordre croissant de coût.
3. Pour chaque arête dans l'ordre trié:
    a. Si l'ajout de l'arête ne crée pas de cycle prématuré
        et n'augmente pas le degré des sommets au-delà de 2,
        ajouter l'arête à la tournée.
4. Répéter jusqu'à ce que toutes les villes soient connectées.
5. Retourner la tournée construite.
```

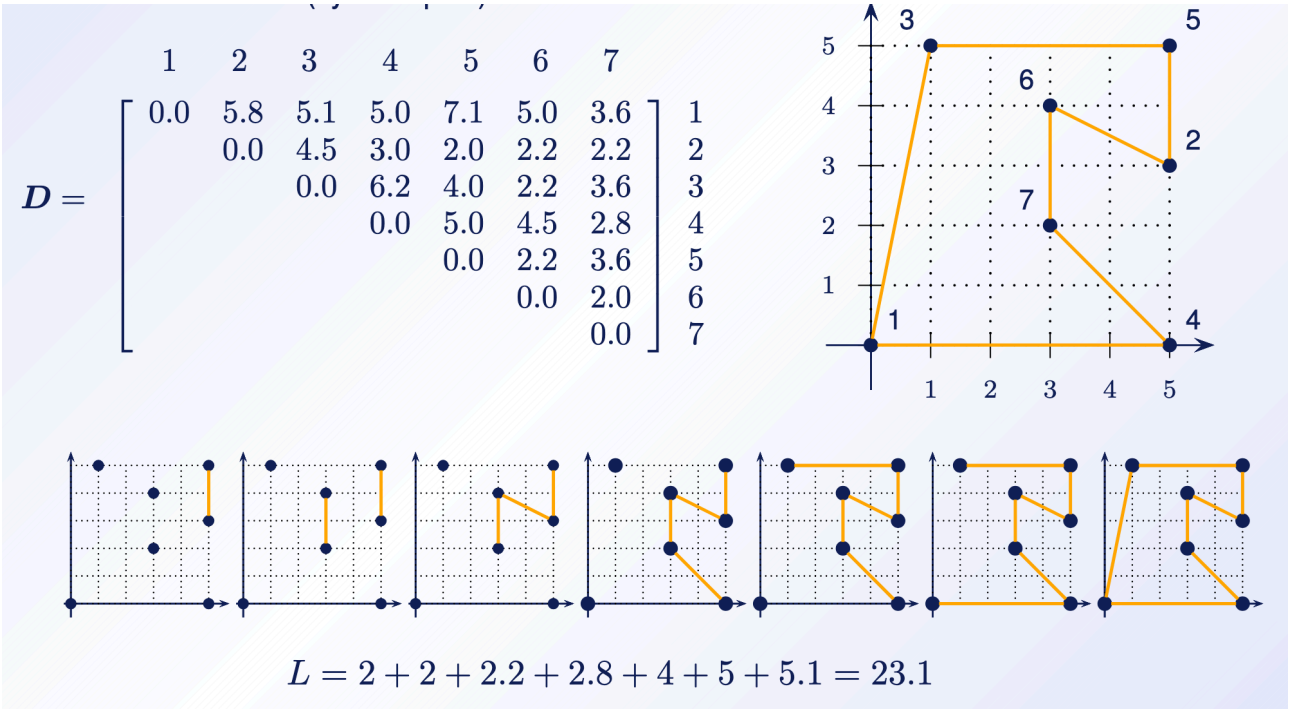

Fig. 6. – Capture des slides du cours – Exemple de l'heuristique gloutonne

3.3 Heuristiques d'insertion (la moins coûteuse)

L'heuristique d'insertion la moins coûteuse pour le TSP consiste à construire une tournée en insérant à chaque étape la ville non encore incluse qui entraîne l'augmentation de coût la plus faible lorsqu'elle est insérée entre deux villes déjà présentes dans la tournée.

3.3.1 Algorithme

```
1. Construire une tournée partielle en choisissant une ville et sa plus proche voisine.
2. Déterminer l'arête {i,j} de la tournée actuelle et calculer le coût d'insertion de la ville
candidate entre i et j.
3. Insérer la ville candidate entre i et j si cela entraîne l'augmentation de coût la plus
faible.
4. Répéter jusqu'à ce que toutes les villes soient incluses dans la tournée.
```

3.4 Heuristique d'insertion (la plus coûteuse)

L'heuristique d'insertion la plus coûteuse pour le TSP consiste à construire une tournée en insérant à chaque étape la ville non encore incluse qui entraîne l'augmentation de coût la plus élevée lorsqu'elle est insérée entre deux villes déjà présentes dans la tournée.

3.4.1 Algorithme

```
1. Construire une tournée partielle en choisissant une ville et sa plus proche voisine.
2. Déterminer l'arête {i,j} de la tournée actuelle et calculer le coût d'insertion de la ville
candidate entre i et j.
3. Insérer la ville candidate entre i et j si cela entraîne l'augmentation de coût la plus
élevée.
4. Répéter jusqu'à ce que toutes les villes soient incluses dans la tournée.
```

3.5 Christofides

L'algorithme de Christofides est une heuristique pour le problème du voyageur de commerce (TSP) qui garantit une solution dont le coût est au plus 1,5 fois le coût optimal, à condition que la distance entre les villes satisfasse l'inégalité triangulaire.

Principe de base: Transformer un cycle eulérien (pas trop long) en un cycle hamiltonien (qui, on l'espère, ne soit pas trop long non plus).

3.5.1 Algorithme

```
1. Déterminer un arbre recouvrat T de poids minimum dans le graphe pondéré complet.
2. Dans el sous-graphe de G induit par les sommets de degré impair dans T, détereminer un
couplage parfait M de poids minimum.
3. Ajouter les arêtes de M à l'arbre optimal T (en dupliquant les arêtes appartenant à M et à T).
Le graphe obtenu a tous ses sommets de degré pair et possède donc un cycle eulérien.
4. Construire la tournée en parcourant ce cycle tout en prenant des raccourcis afin d'éviter de
visiter une ville plus d'une fois.
```

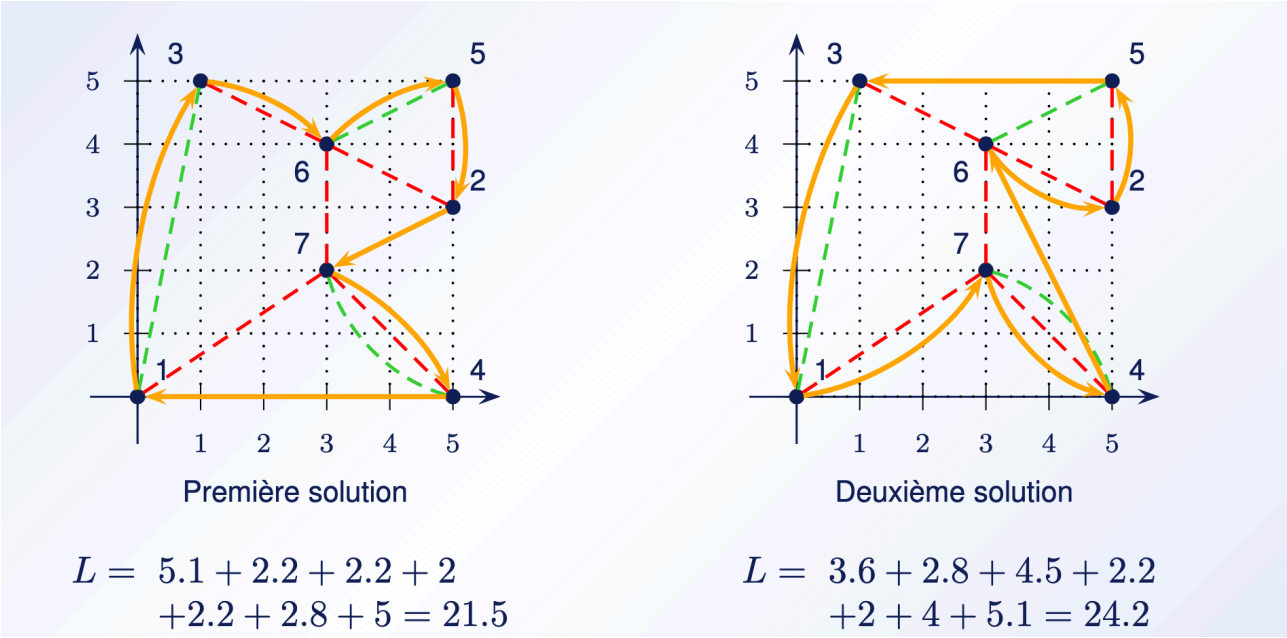

Fig. 7. – Capture des slides du cours – Exemple de l'heuristique de Christofides

3.6 Heuristique « Meilleurs fusions »

L'heuristique des meilleurs fusions pour le TSP consiste à construire une tournée en fusionnant à chaque étape les deux sous-tours dont la fusion entraîne la plus faible augmentation de coût, jusqu'à ce qu'un seul tour reste.

3.6.1 Algorithme

```
1. Initialiser chaque ville comme un sous-tour individuel.
2. Calculer le coût de fusion pour chaque paire de sous-tours.
3. Fusionner la paire de sous-tours avec le coût de fusion le plus faible.
4. Répéter les étapes 2 et 3 jusqu'à ce qu'il ne reste qu'un seul sous-tour.
5. Retourner la tournée construite.
```

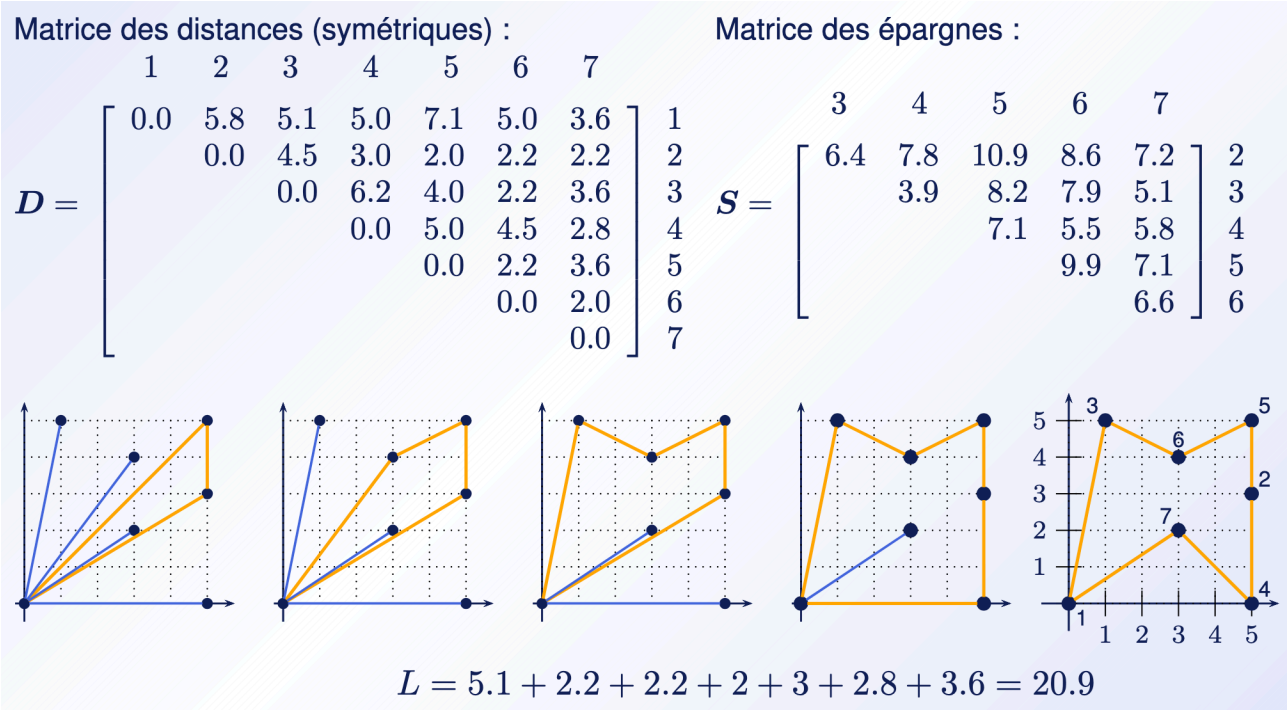

Fig. 8. – Capture des slides du cours – Exemple de l'heuristique des meilleurs fusions

4 Heuristiques d'échanges

4.1 Heuristique d'amélioration

Les heuristiques se basent sur les observations suivantes, valable pour de nombreux problèmes d'optimisation combinatoire:

- Il est souvent facile d'obtenir une solution admissible (éventuellement de mauvaise qualité) à un problème d'optimisation combinatoire donné.
- Il est souvent possible de modifier légèrement et localement une solution tout en conservant son admissibilité.

Les structures de ces techniques sont les suivantes:

1. Générer une solution initiale admissible initiale
2. Modifier localement cette solution pour obtenir une nouvelle solution admissible, meilleure que l'actuelle
3. Répéter l'étape 2 tant qu'on trouve des améliorations possibles

> **i Info**
>
> Une modification locale d'une solution est appelée un **mouvemment**.

4.1.1 Algorithme de descente (structure générale)

```
Répéter
  Parcourir toutes les solutions voisines de la solution actuelle s, autrement dit considérer
  tous les mouvements de M (s) jusqu'à
    Variante 1 : premier voisin améliorant
    Trouver un mouvement m tel que f (s ⊕ m) < f (s)
    Remplacer alors s par s ⊕ m
    Variante 2 : meilleur voisin améliorant
    Trouver le mouvement m qui minimise f (s ⊕ m)
    Remplacer alors s par s ⊕ m si f (s ⊕ m) < f (s)
Tant que l'on trouve au moins un mouvement améliorant.
```

4.2 Heuristique k-opt

L'heuristique k-opt est une méthode d'amélioration pour le problème du voyageur de commerce (TSP) qui consiste à échanger k arêtes dans une tournée pour obtenir une nouvelle tournée de coût inférieur. On répète ce processus jusqu'à ce qu'aucun échange k-opt n'améliore davantage la solution.

En pratique, les heurisitques k-opt les plus utilisées sont les heuristiques 2-opt et 3-opt ou des variantes intermédiaires (2.5-opt ou Or-opt). Des valeurs plus grandes de k n'apportent que de faibles améliorations alors que le coût de calcul augmente rapidement.

4.3 Heuristique 2-opt

L'heuristique 2-opt pour le TSP consiste à améliorer une tournée en supprimant deux arêtes et en reconnectant les segments résultants de manière à réduire la longueur totale de la tournée. Ce processus est répété jusqu'à ce qu'aucun échange 2-opt n'améliore davantage la solution.

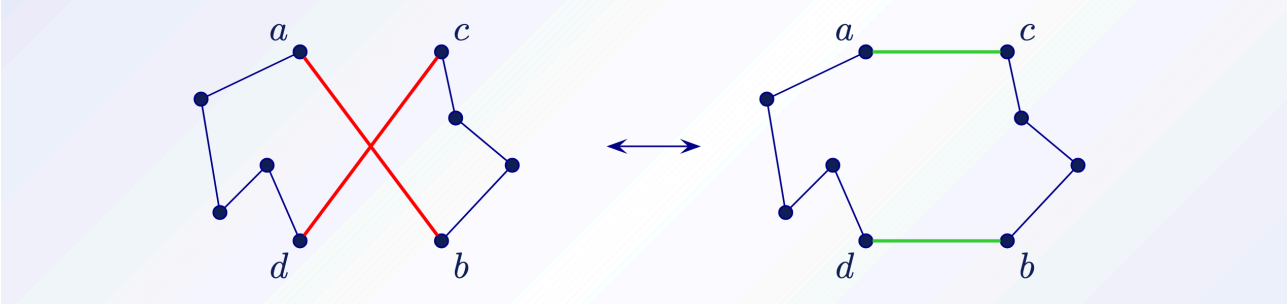

Fig. 9. – Capture des slides du cours – Exemple de l'heuristique 2-opt

> ⚠ **Warning**
>
> Le « **coût** » d'une modification peut être calculé en temps constant mais la modification de la tournée nécessite l'utilisation d'une « bonne » structure de données (une approche directe peut demander un temps $O(n)$ par modification car le sens de parcours entre b et c, ou entre a et d, est inversé après la modification)

4.4 Heuristique 3-opt

L'heuristique 3-opt pour le TSP consiste à améliorer une tournée en supprimant trois arêtes et en reconnectant les segments résultants de manière à réduire la longueur totale de la tournée. Ce processus est répété jusqu'à ce qu'aucun échange 3-opt n'améliore davantage la solution.

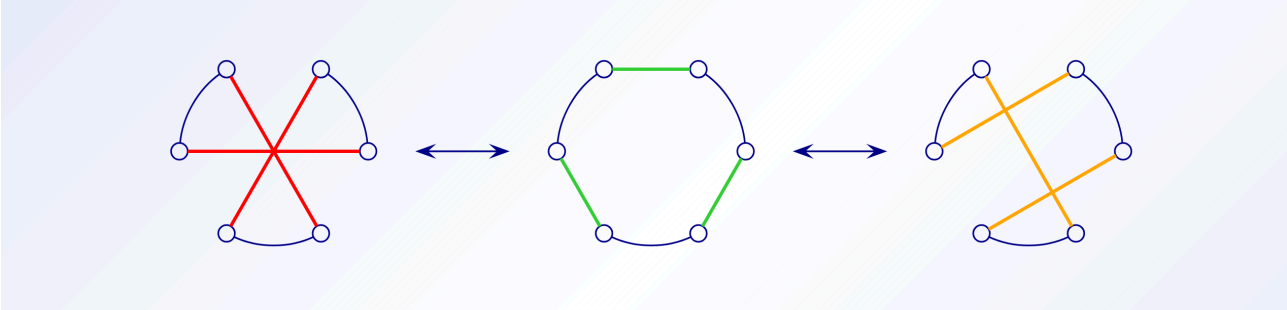

Fig. 10. – Capture des slides du cours – Exemple de l'heuristique 3-opt

5 Métaheuristiques