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.

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

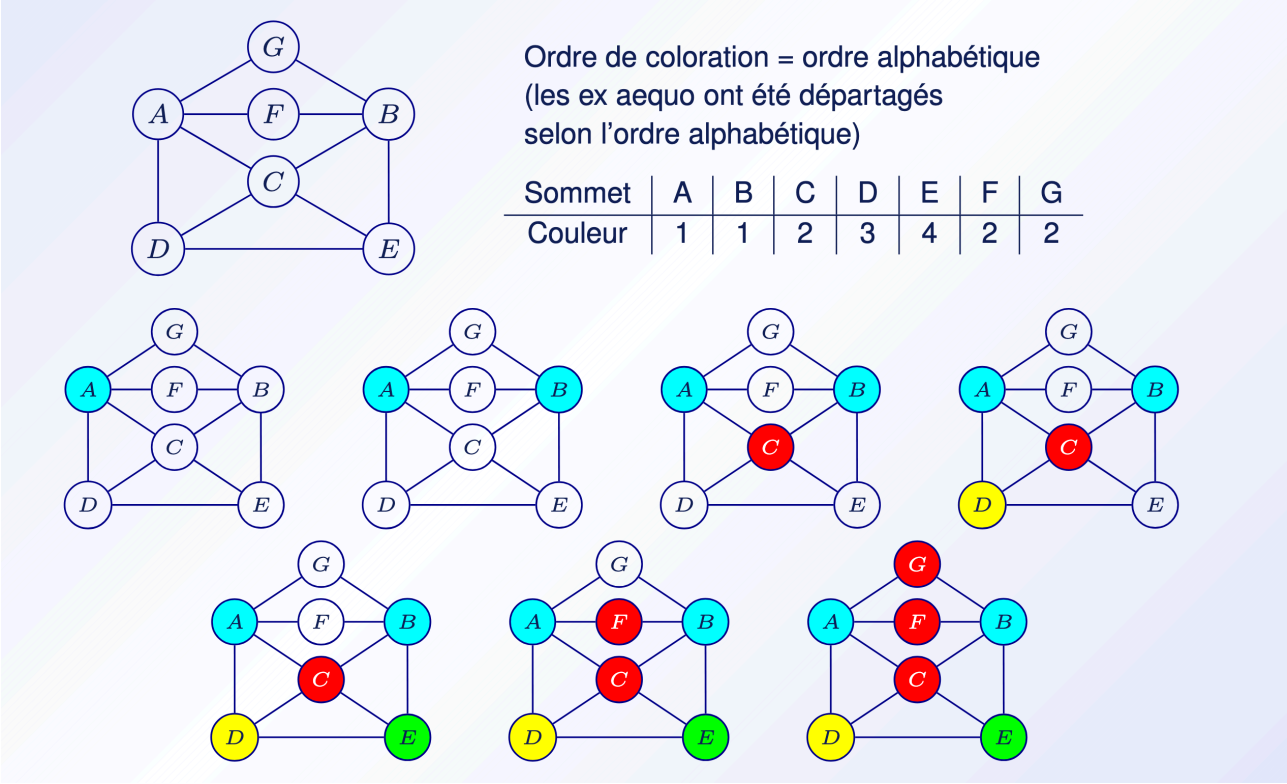

Fig. 1. – Capture des slides du cours – Exemple de la heuristique LF

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

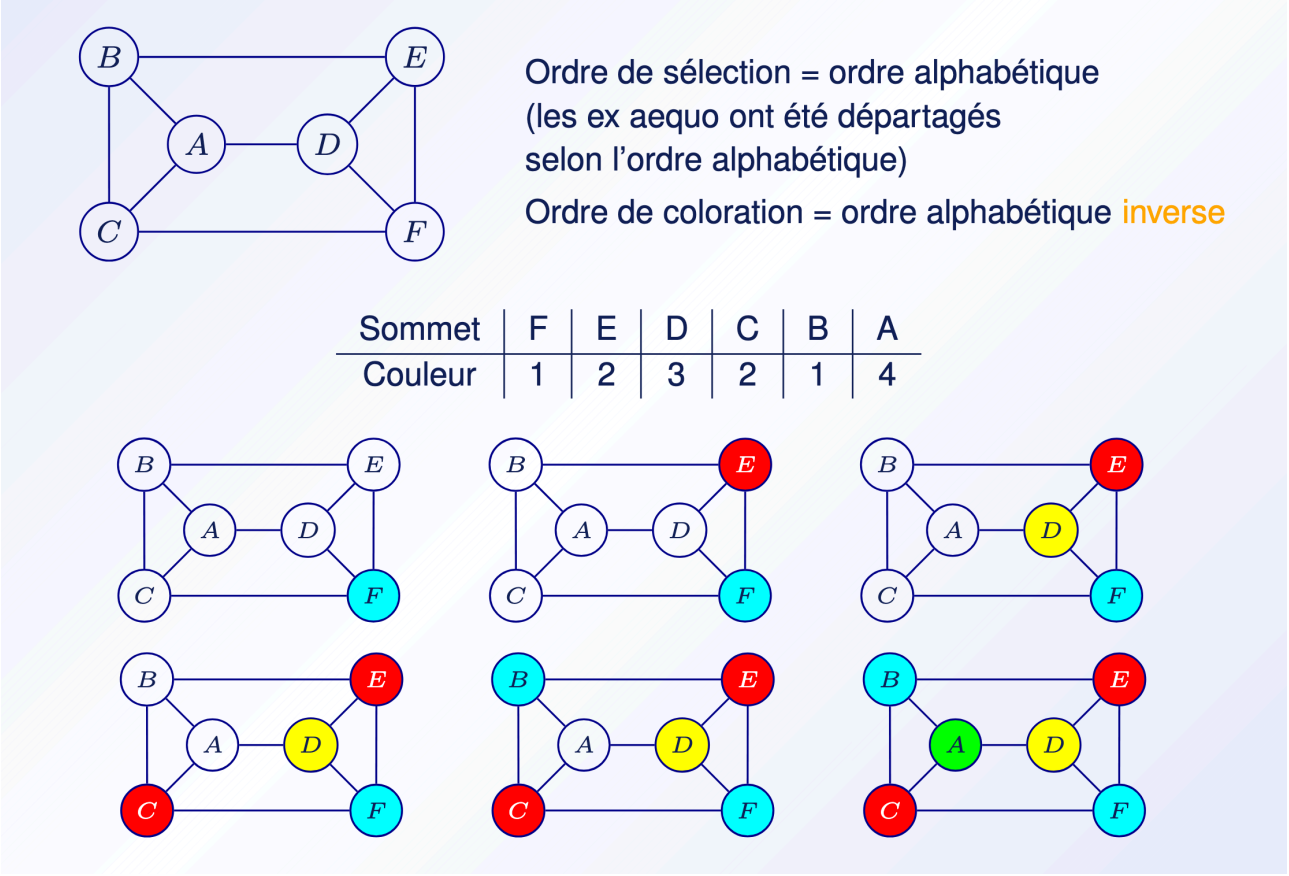

Fig. 2. – Capture des slides du cours – Exemple de la heuristique SL

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

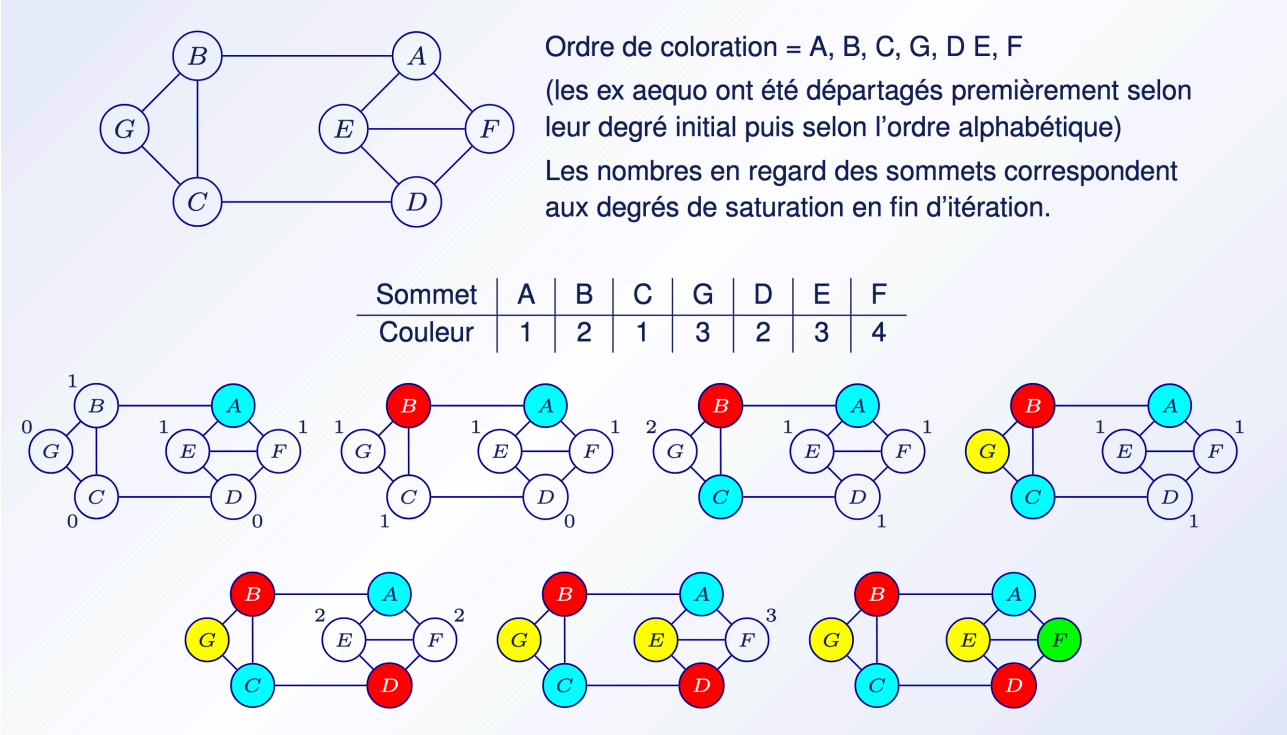

Fig. 3. – Capture des slides du cours – Exemple de la heuristique DSATUR

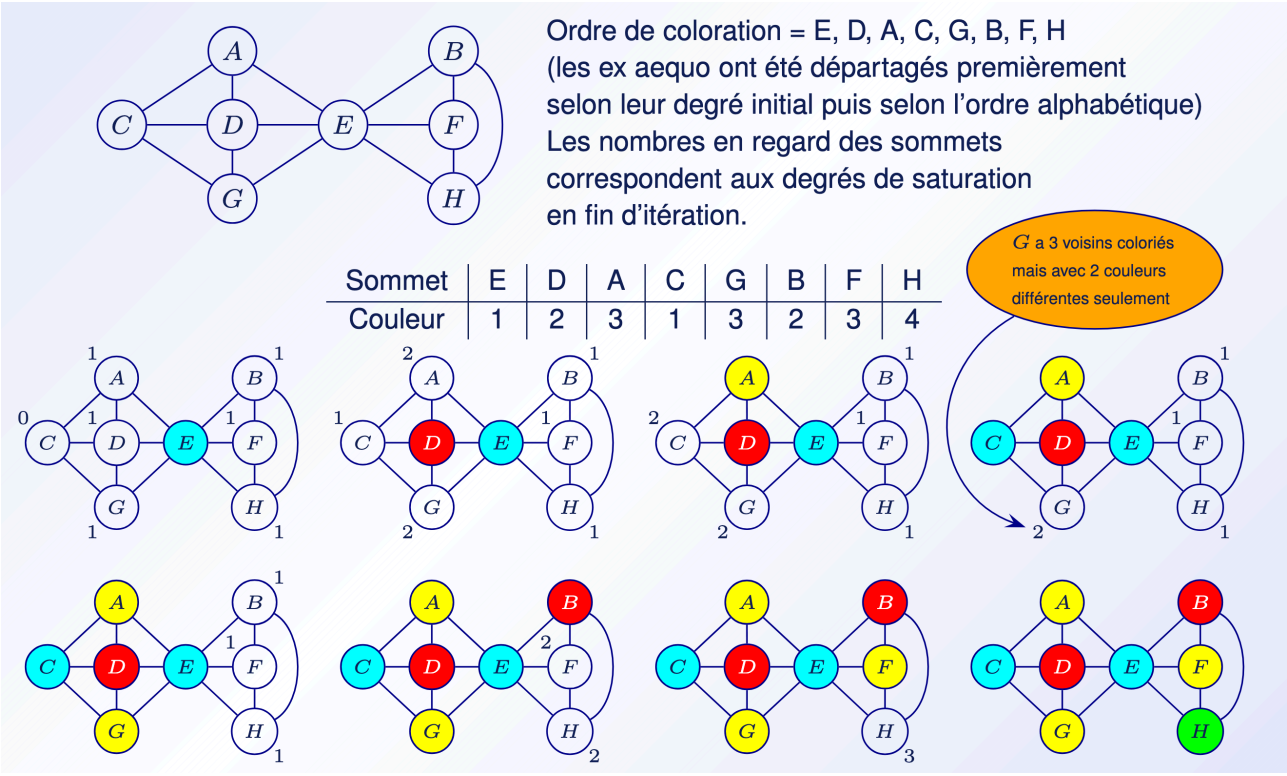

Fig. 4. – Capture des slides du cours – Exemple 2 de la heuristique DSATUR

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

4 Heuristiques d'échanges

5 Métaheuristiques