

# Résolution de Problèmes

## Recherche Locale

Marie Pelleau  
marie.pelleau@univ-cotedazur.fr

Master 1 - Semestre 1

## Algorithme glouton

### Principe

- À chaque étape, on fait un choix, celui qui semble le meilleur à cet instant
- Construit une solution pas à pas
  - sans revenir sur ses décisions
  - en effectuant à chaque étape le choix qui semble le meilleur
  - en espérant obtenir un résultat optimum global
- Approche glouton
  - suivant les problèmes pas de garantie d'optimalité (heuristique gloutonne)
  - peu coûteuse (comparée à une énumération exhaustive)
  - choix intuitif

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

## Recherche locale

### Principe

- On part d'une solution initiale
- À chaque étape, on modifie la solution
  - en essayant d'améliorer la valeur de la fonction objectif
  - en espérant obtenir l'optimum global
- Approche locale
  - suivant les problèmes pas de garantie d'optimalité (heuristique)
  - peu coûteuse

### Solution initiale

- Solution "vide"
- Solution aléatoire
- Solution d'un algorithme glouton

Notes

---

---

---

---

---

---

---

---

---

---

## Recherche locale

### Principe

- On part d'une solution initiale
- À chaque étape, on modifie la solution
  - en essayant d'améliorer la valeur de la fonction objectif
  - en espérant obtenir l'optimum global
- Approche locale
  - suivant les problèmes pas de garantie d'optimalité (heuristique)
  - peu coûteuse

### Modifications

- Modifie la valeur d'une variable
- Échange la valeur de deux variables

Notes

---

---

---

---

---

---

---

---

---

---

## Le sac-à-doc (knapsack)

### Description

On a :

- Un Sac dans lequel on peut mettre un poids limité
- Un ensemble d'objets, chaque objet  $o_i$  a
  - Un poids :  $p_i$
  - Une valeur :  $v_i$

Quels sont les objets que l'on doit prendre pour maximizer la valeur transportée tout en respectant la contrainte de poids ?

- La somme des valeurs des objets pris est maximale
- La somme des poids des objets pris est  $\leq$  poidsmax du sac

## Le sac-à-doc (knapsack)

### Les variables

- On associe à chaque objet une variable 0-1 (elle ne prend que les valeurs 0 ou 1)
- C'est une variable d'appartenance au sac à dos
- Si l'objet est pris alors la variable vaut 1 sinon elle vaut 0

Notes

---

---

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

---

---

## Le sac-à-doc (knapsack)

### Modèle

- La valeur d'un objet et son poids sont des données, donc pour l'objet  $o_i$  on a la valeur  $v_i$  et le poids  $p_i$
- La variable d'appartenance au sac est  $x_i$
- Le poids maximum du sac est  $W$

### Les contraintes

- $\max \sum_{i=1}^n v_i x_i$  l'objectif
- $\sum_{i=1}^n p_i x_i \leq W$  somme des poids inférieure ou égale au poids maximal

Notes

---

---

---

---

---

---

---

---

## Le sac-à-doc (knapsack)

### Solution initiale

- Solution "vide" : sac à dos vide  $\Rightarrow$  fonction objectif 0
- Solution aléatoire : sac à dos aléatoire  $\Rightarrow$  il faut vérifier que c'est une solution
- Solution d'un algorithme glouton

### Modifications

- Ajoute un élément au sac à dos  $\Rightarrow$  si la capacité max n'est pas dépassée
- Supprime un élément du sac à dos

Notes

---

---

---

---

---

---

---

---

## Hitting-set : Recouvrement (set cover)

### Description

- Un interrupteur est relié à certaines ampoules
- Si on appuie sur l'interrupteur alors on allume toutes les ampoules reliées
- **Question** : sur combien d'interrupteur au minimum doit-on appuyer pour allumer toutes les ampoules ?

## Hitting-set : Recouvrement (set cover)

### Solution initiale

- Solution "vide" : tous les interrupteurs allumés  $\Rightarrow$  fonction objectif nombre d'interrupteurs
- Solution aléatoire : position des interrupteurs aléatoire  $\Rightarrow$  il faut vérifier que c'est une solution
- Solution d'un algorithme glouton

### Modifications

- Allume un interrupteur
- Éteint un interrupteur  $\Rightarrow$  si toutes les ampoules restent allumées

Notes

---

---

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

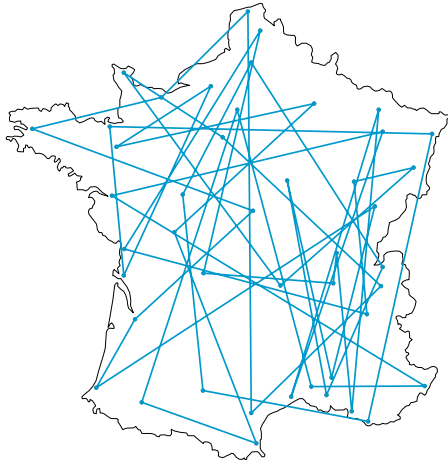
---

---

---

---

## TSP



## Solution initiale

- Les villes par ordre alphabétique

Notes

---

---

---

---

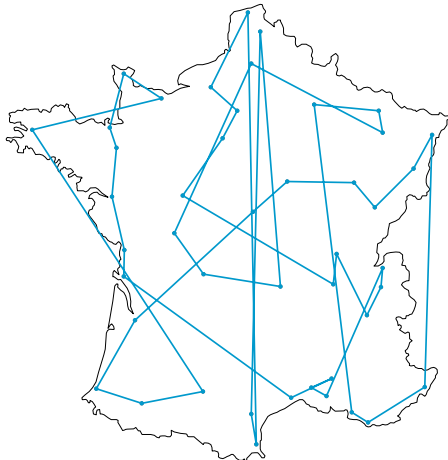
---

---

---

---

## TSP



## Solution initiale

- Les villes par ordre alphabétique
- Les villes dans un ordre aléatoire

Notes

---

---

---

---

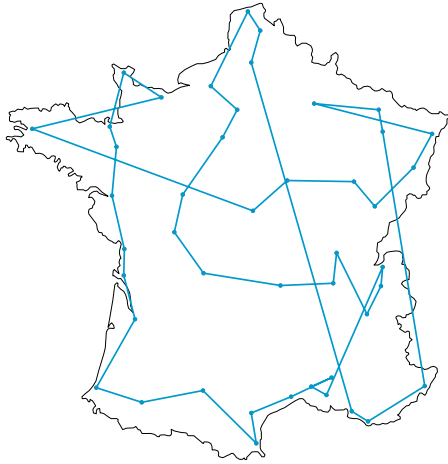
---

---

---

---

## TSP



## Solution initiale

- Les villes par ordre alphabétique
- Les villes dans un ordre aléatoire
- Solution d'un algorithme glouton

Notes

---

---

---

---

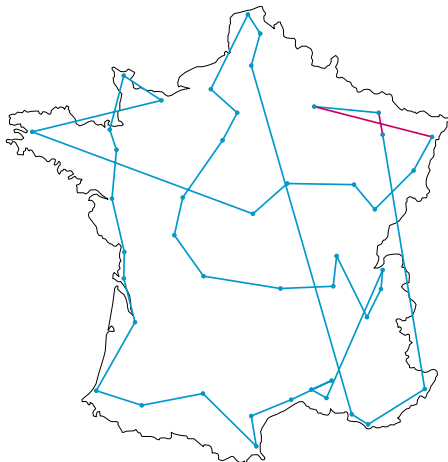
---

---

---

---

## TSP



## Modifications

- $k$ -opt
  - $k = 2$
  - $k = 3$

Notes

---

---

---

---

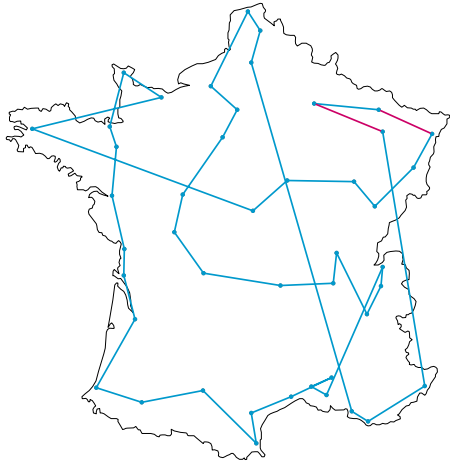
---

---

---

---

## TSP



## Modifications

- $k$ -opt
- $k = 2$
- $k = 3$

Notes

---

---

---

---

---

---

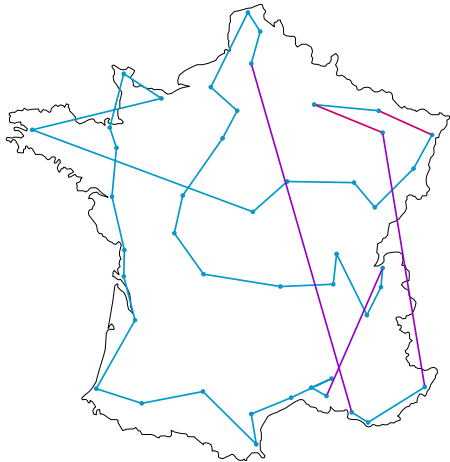
---

---

---

---

## TSP



## Modifications

- $k$ -opt
- $k = 2$
- $k = 3$

Notes

---

---

---

---

---

---

---

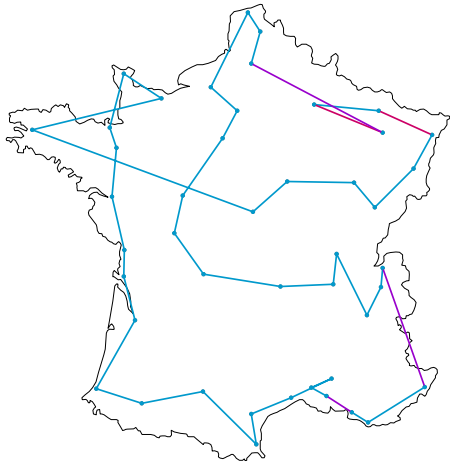
---

---

---



## TSP



## Modifications

- $k$ -opt
  - $k = 2$
  - $k = 3$

Notes

---

---

---

---

---

---

---

---

## Recherche locale

## Principe

- On part d'une solution initiale
- À chaque étape, on modifie la solution  $\Rightarrow$  notion de voisinage

## Voisinage

Pour une solution, l'ensemble des solutions à une modification près

Notes

---

---

---

---

---

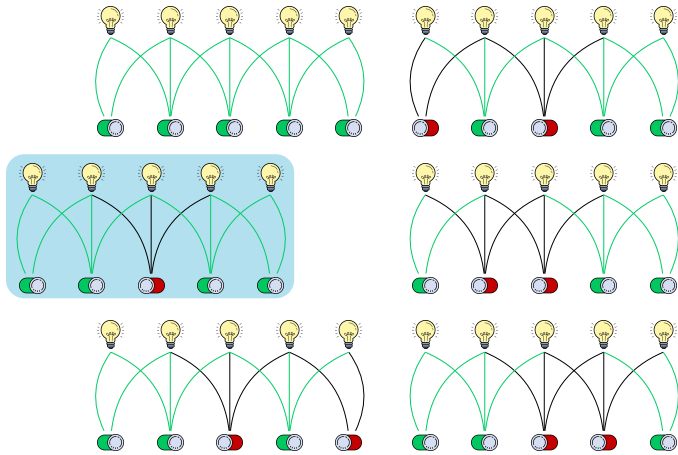
---

---

---

## Hitting-set : Recouvrement (set cover)

### Voisinage



Notes

---

---

---

---

---

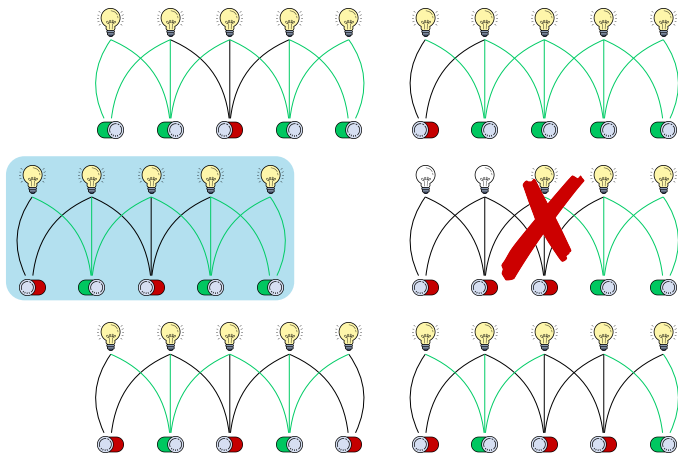
---

---

---

## Hitting-set : Recouvrement (set cover)

### Voisinage



Notes

---

---

---

---

---

---

---

---

## Recherche locale

### Principe

- On part d'une solution initiale
- À chaque étape, on modifie la solution (on choisit un voisin)

### Quel voisin choisir ?

- Aléatoirement
- Le meilleur
- Un parmi les meilleurs

## Plan

- 1 Marche aléatoire
- 2 Algorithme de la descente
- 3 Restarts
- 4 Recherche Tabou
- 5 Recherche locale
- 6 Constraint Based Local Search

### Notes

---

---

---

---

---

---

---

---

### Notes

---

---

---

---

---

---

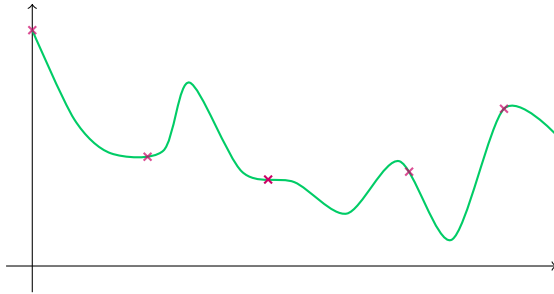
---

---

## Marche aléatoire

### Principe

- On part d'une solution initiale
- À chaque étape, on modifie **aléatoirement** la solution



Notes

---

---

---

---

---

---

---

---

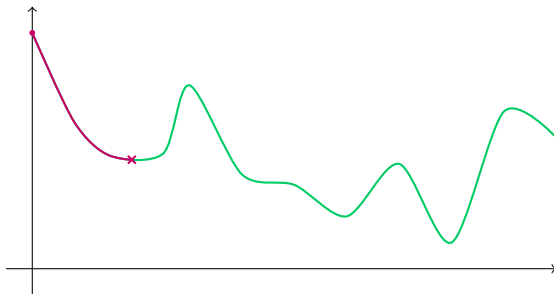
---

---

## Algorithme de la descente

### Principe

- On part d'une solution initiale
- À chaque étape, on se déplace vers une solution du voisinage **améliorant strictement** l'objectif



### Inconvénients

On peut rester bloquer dans des minimum locaux

Notes

---

---

---

---

---

---

---

---

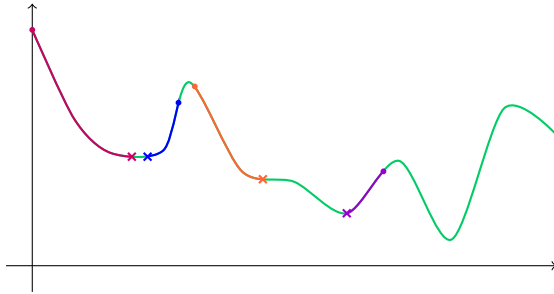
---

---

## Algorithme de la descente

### Principe

- On part d'une solution initiale
- À chaque étape, on se déplace vers une solution du voisinage **améliorant strictement** l'objectif



### Restarts

On recommence à partir d'une autre solution

Notes

---

---

---

---

---

---

---

---

---

---

## Recherche locale

### Restarts

- Solution aléatoire
- Solution "vide", dans laquelle on fixe un certain pourcentage de variables comme dans la meilleure solution trouvée jusqu'ici
  - 5%, 10%, 20%

**Large Neighborhood Search** (LNS) [Shaw, 1998]

### Pas d'amélioration

- On se déplace vers une solution du voisinage **sans améliorer** l'objectif  
⇒ Il ne faut pas être un poisson rouge

Notes

---

---

---

---

---

---

---

---

---

---

## Recherche Tabou [Glover, 1986]

### Principe

- On part d'une solution  $s$
- On se déplace vers **la meilleure** solution du voisinage qui ne soit pas **interdite**
- On ajoute  $s$  aux solutions interdites pour les  $m$  itérations suivantes

### Mémoire

- Interdire des solutions peut être coûteux en mémoire
- À la place on interdit des mouvements

### Critère d'aspiration

On peut accepter un mouvement tabou s'il permet d'obtenir une **meilleure** solution que la meilleure solution connue jusqu'ici

Notes

---

---

---

---

---

---

---

---

## Taille de la liste taboue

- Si  $m$  trop faible, **intensification** trop forte  $\Rightarrow$  blocage de la recherche autour d'un optimum local
- Si  $m$  trop grand, **diversification** trop forte  $\Rightarrow$  risque de rater des solutions

La longueur optimale de la liste varie

- d'un problème à l'autre
- d'une instance à l'autre d'un même problème
- au cours de la résolution d'une même instance

[Battiti, Protasi 2001] : adapter cette longueur dynamiquement

- Besoin de diversification  $\Rightarrow$  augmenter  $m$
- Besoin de d'intensification  $\Rightarrow$  diminuer  $m$

Notes

---

---

---

---

---

---

---

---

## Recherche locale

### Principe

- On part d'une solution initiale
- À chaque étape, on modifie la solution
  - en essayant d'améliorer la valeur de la fonction objectif
  - en espérant obtenir l'optimum global
- Approche locale
  - suivant les problèmes pas de garantie d'optimalité (heuristique)
  - peu coûteuse

### Remarque

- Cela suppose qu'il existe une fonction objectif
- Comment faire s'il n'en existe pas ?

Notes

---

---

---

---

---

---

---

---

---

---

## Constraint Based Local Search

### Principe

- Étant donné un problème sous la forme
  - $\mathcal{V} = \{v_1, \dots, v_n\}$  : variables
  - $\mathcal{D} = \{D_1, \dots, D_n\}$  : domaines
  - $\mathcal{C} = \{C_1, \dots, C_p\}$  : contraintes
- Fonction objectif à minimiser : nombre de contraintes non satisfaites

### Intuition

- Recherche guidée par la structure du problème
  - les contraintes donnent de la structure au problème et les variables les lient ensemble
- Tout type de contraintes peut être utilisé

Notes

---

---

---

---

---

---

---

---

---

---

## Constraint Based Local Search

### N-reines

- Sur un échiquier de  $n \times n$
- Placer  $n$  reines de telle sorte qu'aucune reine ne puisse en capturer une autre

### Formulation

- $l_i$  : colonne de la reine sur la ligne  $i$
- $l_i \neq l_j$
- $l_i + i \neq l_j + j$  (diagonale montante)
- $l_i - i \neq l_j - j$  (diagonale descendante)

### Fonction objectif

- Nombre de contraintes non satisfaites

Notes

---

---

---

---

---

---

---

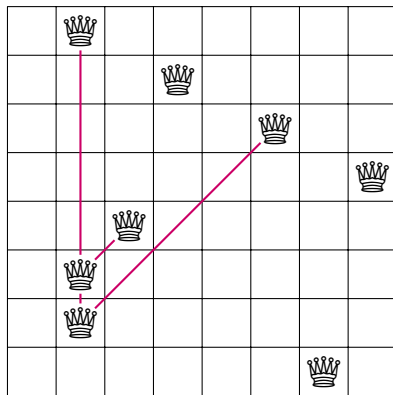
---

---

---

## Constraint Based Local Search

### Formulation



Fonction objectif : 4

Notes

---

---

---

---

---

---

---

---

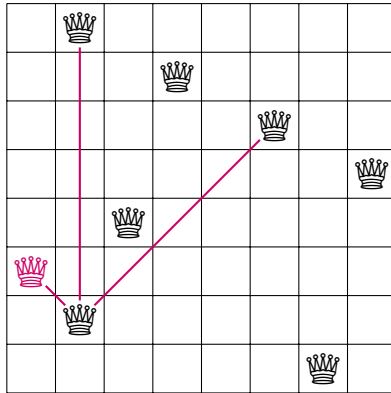
---

---



## Constraint Based Local Search

### Formulation



Fonction objectif : 3

Notes

---

---

---

---

---

---

---

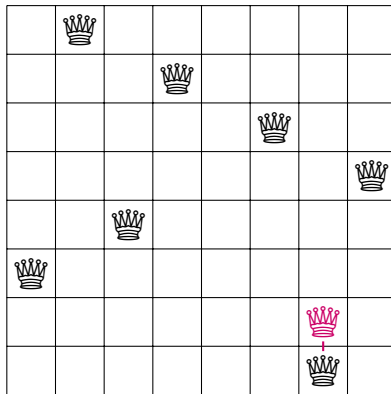
---

---

---

## Constraint Based Local Search

### Formulation



Fonction objectif : 1

Notes

---

---

---

---

---

---

---

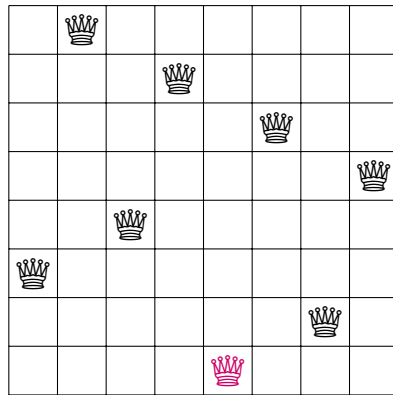
---

---

---

# Constraint Based Local Search

## Formulation



Fonction objectif : 0

## Notes

---

---

---

---

---

---

---

---

---

---

## Notes

---

---

---

---

---

---

---

---

---

---