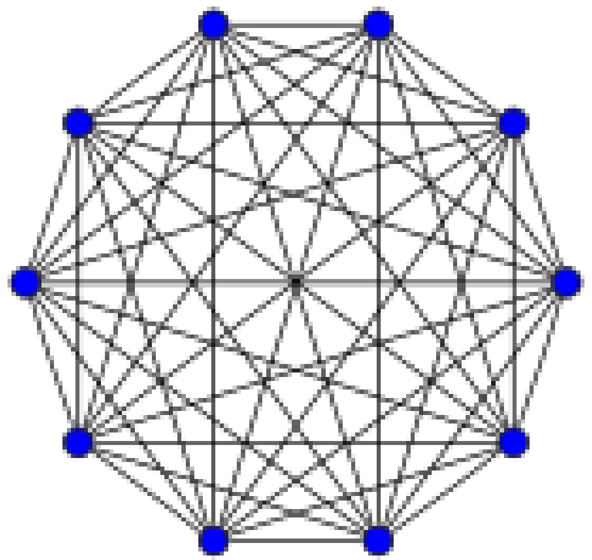
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
RENNES

The Travelling Salesman Problem

LOMMEL Mathias, PERRIN Aglaé, THOMAS Julie, EL FAKIR Maryam, DEMOLLIERE Bastien

INTRODUCTION

Trouver le plus court chemin, en visitant toutes les villes une et une seule fois



Minimisation sous contraintes

Pour n villes : n! chemins possibles

Pour le résoudre, différentes stratégies :

1

PROGRAMMATION DYNAMIQUE

Algorithme de Held-Karp

2

PROGRAMMATION LINÉAIRE EN NOMBRE ENTIER

Méthodes de MTZ et DFJ

3

SOLUTION APPROCHÉE

Lin-Kernighan Heuristic
Nearest Neighbor
Ant colony

Données du problème

- Soit G=(S,A) : graphe complet, non orienté
- |S| = n nombre de villes à visiter
- $C \in \mathbb{M}_n(\mathbb{R})$ la matrice d'adjacence du graphe, avec c_{ij} le coût d'aller de la ville i vers la ville j

  $c_{ij} \neq 0, \forall i \in \{1, \ldots, n\}, i \neq j$

  $c_{ij} = c_{ji}$ car le graphe est symétrique

Objectif :

Trouver le cycle Hamiltonien de coût minimum

Algorithme de Held-Karp

Soit $\tilde{S}$ un sous ensemble de villes, de taille k, $\tilde{S} \subseteq \{2, \ldots, n\}$

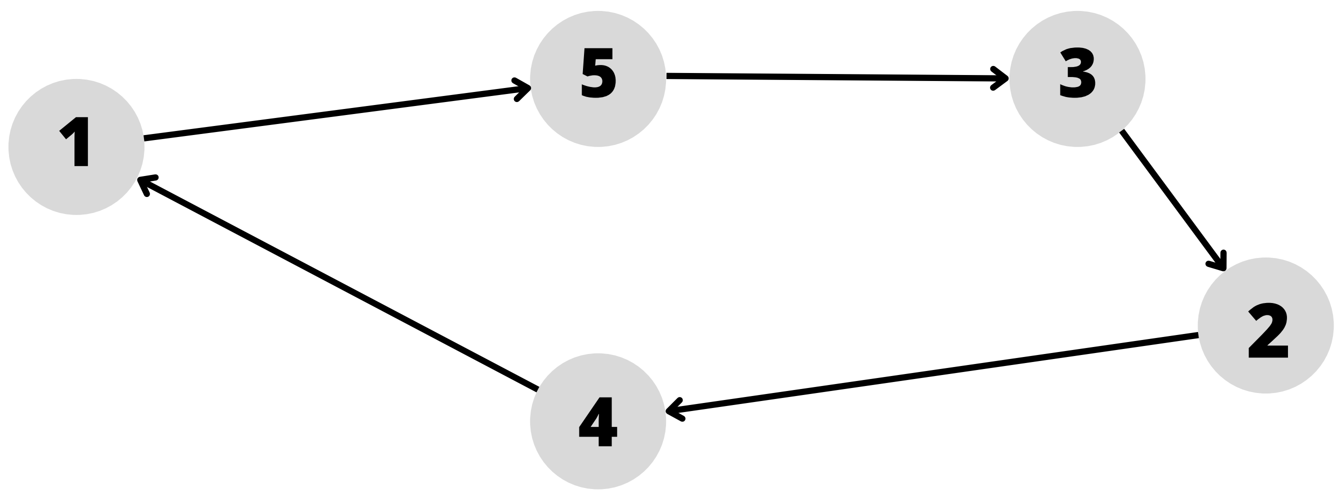
$$g(\tilde{S}, e) = \min_{1 \leq i \leq k} g(\tilde{S} \setminus s_i, s_i) + d(s_i, e)$$

Meilleure distance entre la ville 1 et la ville e en passant par toutes les villes de $\tilde{S}$

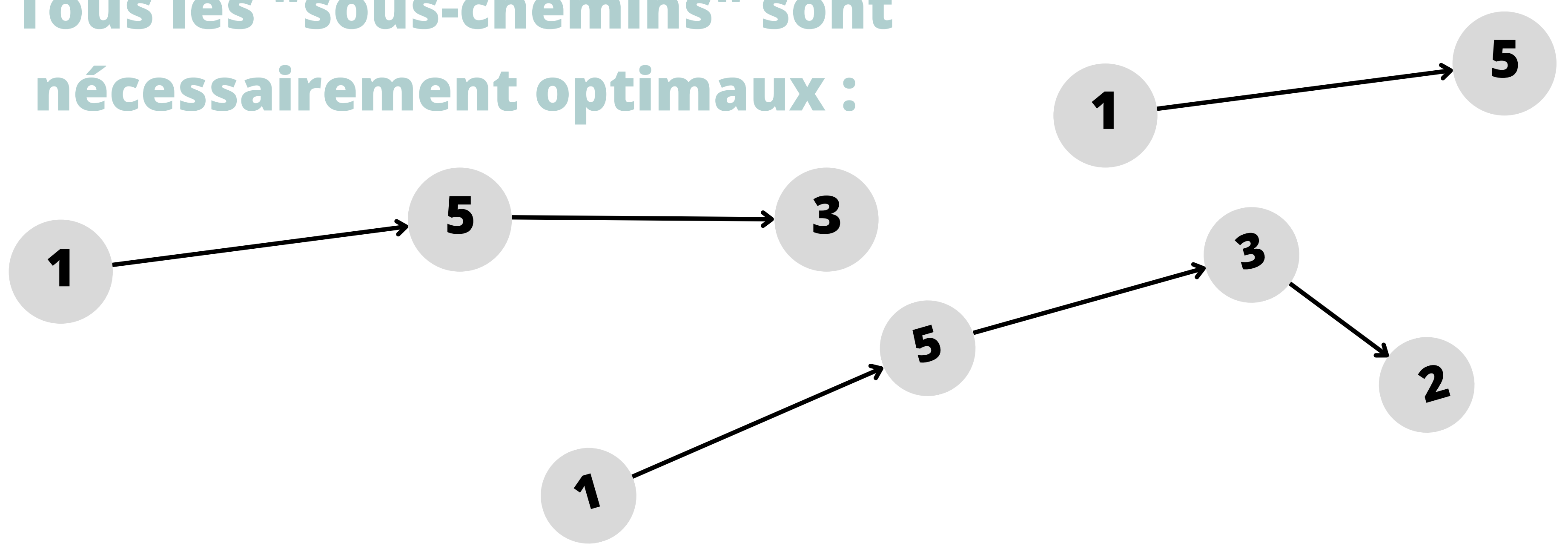
A chaque étape, on utilise le résultat précédent

Distance entre la ville s_i et la ville *e*

Circuit optimal :



Tous les "sous-chemins" sont nécessairement optimaux :



Algorithme de Held-Karp

Nombre de sous-ensembles $\tilde{S}$ de taille k ($|\tilde{S}|$ =k) possibles : C_{n-1}^{k}

Il reste $n-1-k$ valeurs qui ne sont pas dans $\tilde{S}$

$\longrightarrow$ $n-1-k$ **possibilités de sommets d'arrivée**

Complexité en espace :
$$\Theta(2^n n)$$

Calcul de $g(\tilde{S}, e)$ (à e et $\tilde{S}$ fixés) : k calculs intermédiaires à faire avant de trouver le minimum

On fait la somme sur toutes les tailles k

Complexité en temps : $\Theta(2^n n^2)$

n = 17

```
Instance   : gr17.tsp
Method     : hk
Solve time : 0:00:01.934659
Tour cost  : 2085.0
```

2 secondes

n = 21

```
Instance   : gr21.tsp
Method     : hk
Solve time : 0:00:56.941296
Tour cost  : 2707.0
```

57 secondes

n = 24

```
Instance   : gr24.tsp
Method     : hk
Solve time : 0:14:16.748164
Tour cost  : 1272.0
```

14minutes

Formulation du problème

Soient les variables $x_{ij} = 1$ si on quitte la ville i pour aller dans la ville j

= 0 sinon

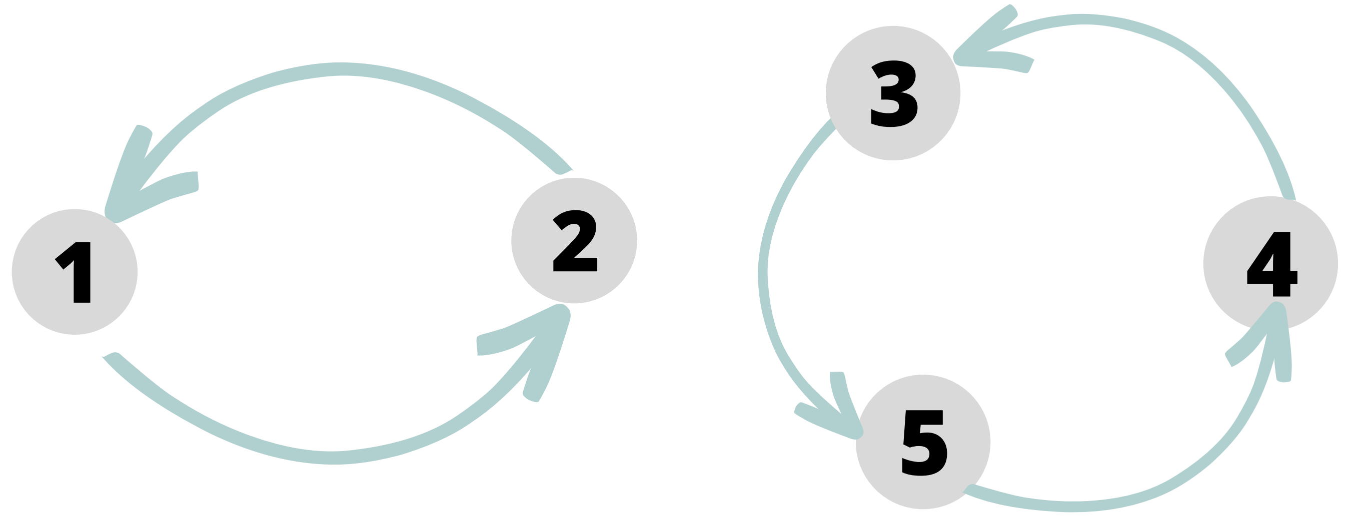
La fonction à minimiser est alors : $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$, sous les contraintes :

$$\sum_{i \in S} x_{ij} = 1 \quad \forall j \in S \qquad\qquad \sum_{j \in S} x_{ij} = 1 \quad \forall i \in S$$

On quitte i une seule fois

On visite j une seule fois

Soient {1,2,3,4,5} un ensemble de villes à visiter :



Ce chemin respecte les contraintes mais ne répond pas au problème

L'algorithme de MTZ

On introduit de nouvelles variables : $u_i, i \in \{1, \ldots n\}$ $\quad u_i \geq 0$

$\longrightarrow$ permettent de classer les villes en fonction de leur ordre de visite

$\longrightarrow$ si $x_{ij} = 1, u_j \geq u_i + 1$
si $x_{ij} = 0$, pas de contraintes sur u_i et u_j

Contrainte supplémentaire : $u_i - u_j + n * x_{ij} \leq n - 1 \quad \forall i, j \in \{2, \ldots n\}$

$\longrightarrow$ élimine les cycles

L'algorithme de DFJ

Pour chaque sous ensemble de sommet, on limite le nombre d'arcs possibles :

$$\sum_{i \in \tilde{S}} \sum_{j \in \tilde{S}, j \neq i} x_{ij} \leq |\tilde{S}| - 1$$

$\forall \tilde{S} \subsetneq \{1, \ldots, n\}$ (pour tous les sous ensembles stricts)

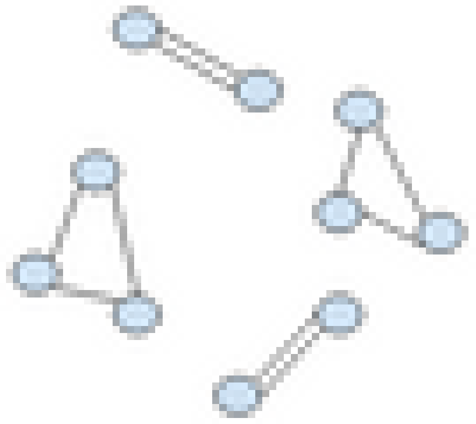
Cette condition garantit qu'aucun sous-ensemble ne peut former un sous-tour : solution connexe garantie.

Problème : nombre exponentiel de contraintes possibles

Solution : algorithme itératif, ajoutant des contraintes à chaque solution trouvée

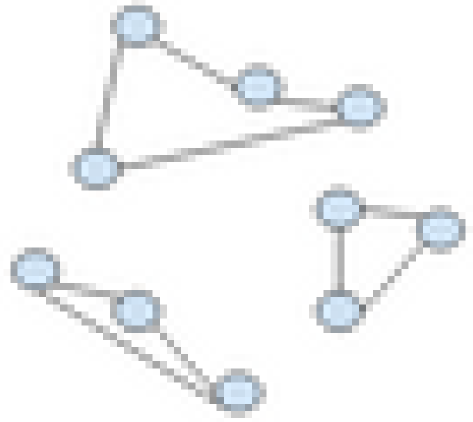
L'algorithme de DFJ

Première solution :



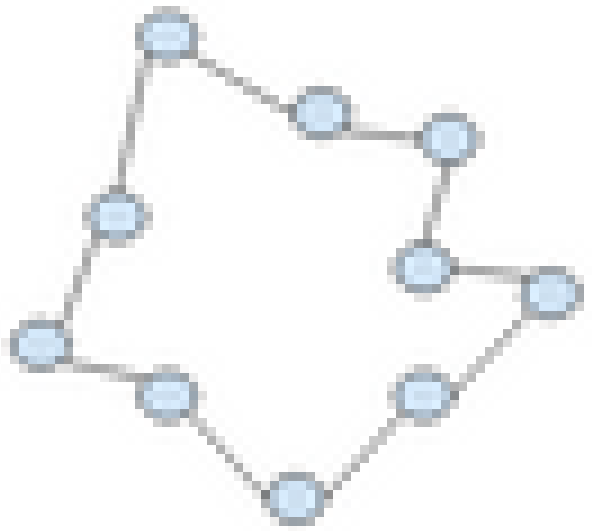
On ajoute les contraintes
présentées précédemment

Seconde solution :



On itère de cette manière jusqu'à obtenir
une solution "en un morceau"

...



Comparaison des 2 méthodes

MTZ

- $u_1 = 0 \rightarrow 1 \text{ contrainte}$
- $2 \leq u_i \leq n \rightarrow 2(n-1) \text{ contraintes}$
- $u_i - u_j + n * x_{ij} \leq n - 1$
  $\longrightarrow \quad (n-1)(n-2) \text{ contraintes}$

$1 + 2(n-1) + (n-1)(n-2)$ contraintes

+ création de n variables supplémentaires

DFJ

- 2^n sous ensembles possibles au total (dans le pire des cas)
- Or, il y a n sous ensembles avec 1 seul sommet
- et les cas $\left.\begin{array}{l} \tilde{S} = S \\ \tilde{S} = \emptyset \end{array}\right\}$ (2 cas) ne sont pas considérés non plus

$2^n - n - 2$ contraintes (au pire)

Comparaison des 2 méthodes

MTZ		DFJ

MTZ

```
Instance    : gr17.tsp
Method      : mtz
Solve time  : 0:00:16.443599
Tour cost   : (1, 'Optimal', 2085.0)
```

```
Instance    : gr21.tsp
Method      : mtz
Solve time  : 0:00:00.575147
Tour cost   : (1, 'Optimal', 2707.0)
```

```
Instance    : gr24.tsp
Method      : mtz
Solve time  : 0:00:00.814952
Tour cost   : (1, 'Optimal', 1272.0)
```

16 s | 0.32 s

≈

DFJ

```
Instance    : gr17.tsp
Method      : dfj
Solve time  : 0:00:00.320380
Tour cost   : (1, 'Optimal', 2085.0)
```

```
Instance    : gr21.tsp
Method      : dfj
Solve time  : 0:00:00.109643
Tour cost   : (1, 'Optimal', 2707.0)
```

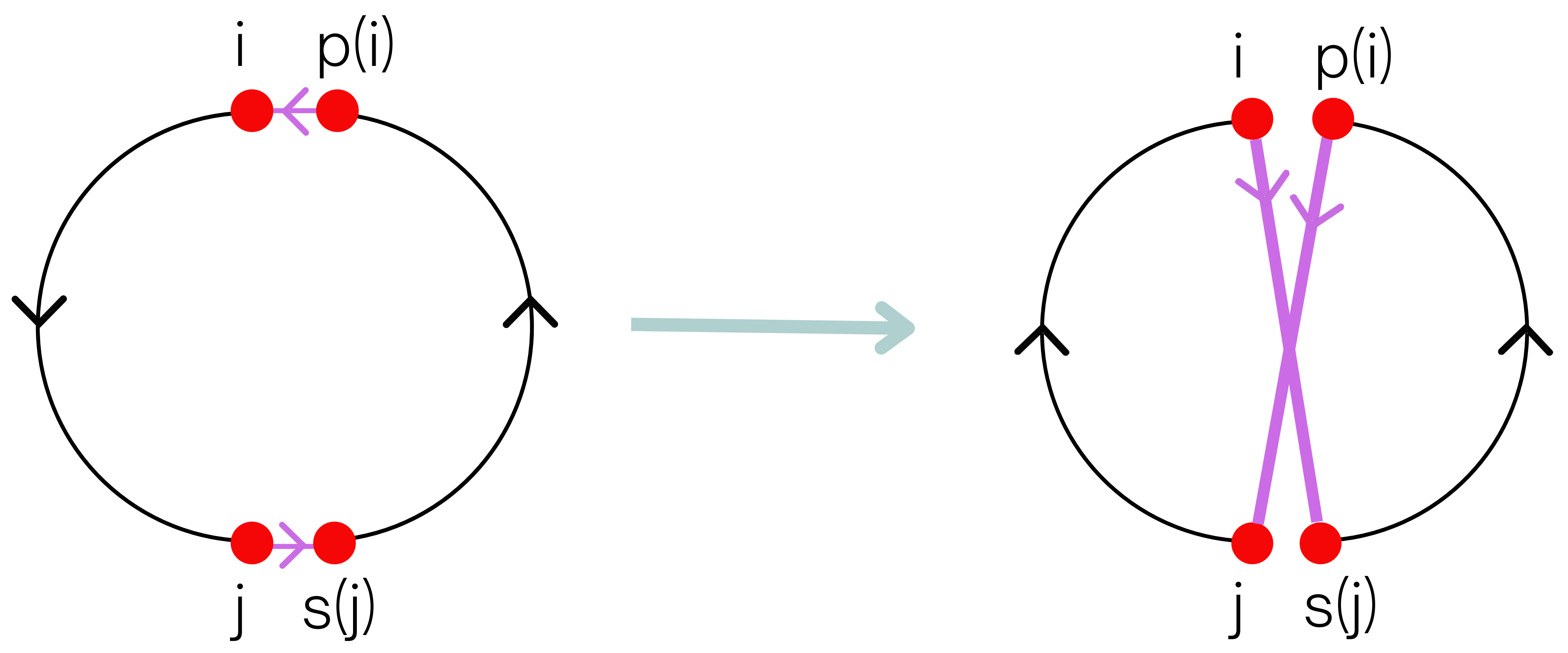
```
Instance    : gr24.tsp
Method      : dfj
Solve time  : 0:00:00.459913
Tour cost   : (1, 'Optimal', 1272.0)
```

3 - SOLUTION APPROCHÉE

Lin-Kernighan Heuristic

Initialisation d'une première route ordonnée aléatoirement

Algorithme en 2-opt :



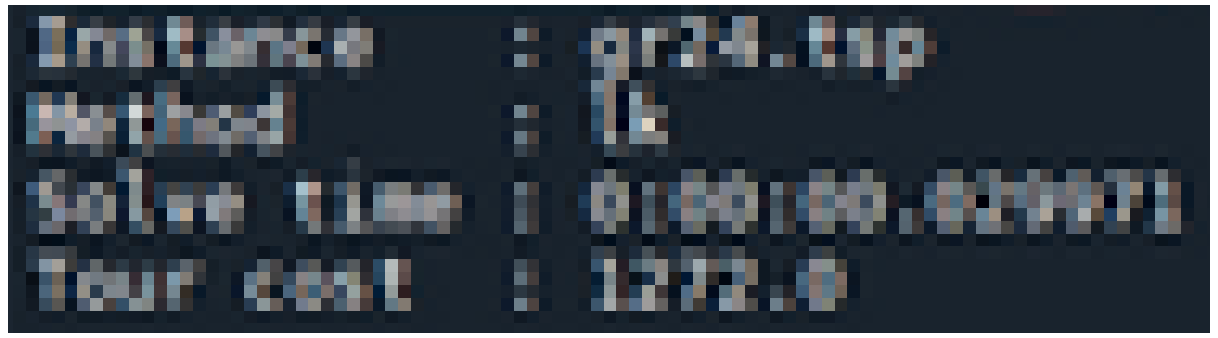
Principe : échanger 2 arcs dans le circuit et voir si le coût total est diminué

- on fixe i ∈ {1,...,n-2} = sommet d'arrivée du premier arc
- on fixe j ∈ {i+1,...,n-1} = sommet de départ du deuxième arc
- on crée la nouvelle route : o→...→p(i)→j→...→i→s(j)→...→o
- on calcule son coût qu'on compare au coût à battre

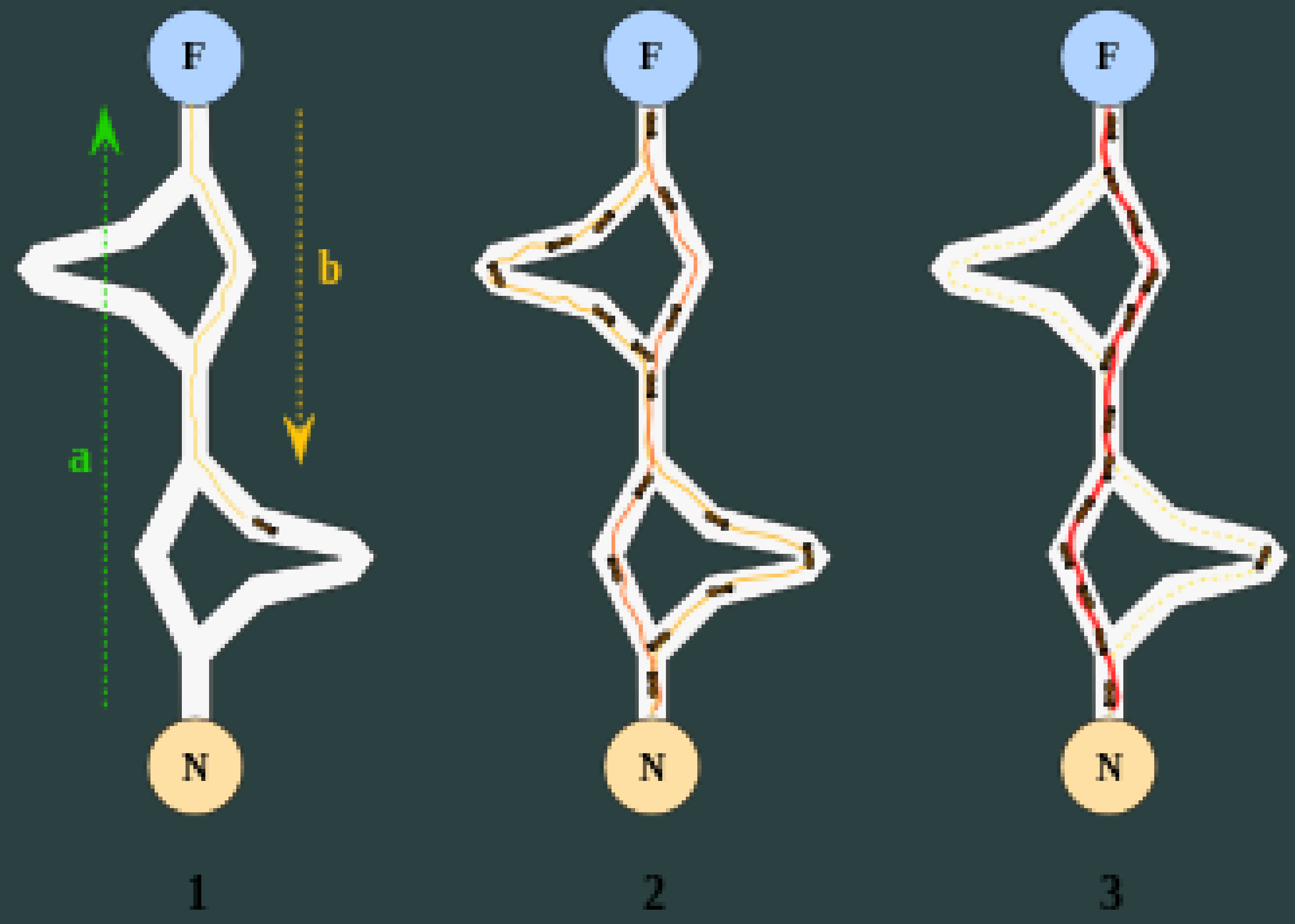
Lin-Kernighan Heuristic

Complexité en temps : PLS-complete
(Polynomial Local Search)

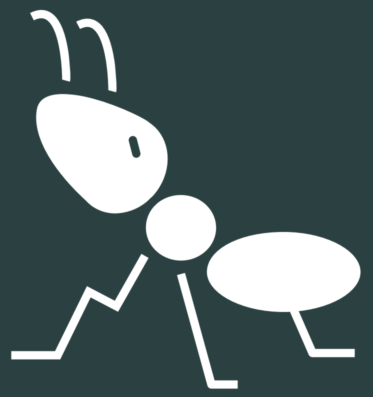
Résultats variables, parfois exacts, toujours proches de la solution



Ant colony algorithm



Principe de l'algorithme :



- Envoi de groupes de fourmis
- Chaque fourmi dépose des phéromones sur les arcs empruntés
- Les fourmis auront tendance à se déplacer à travers les chemins les plus phéromonés

Ant colony algorithm

p(x,y) : poids associé à l'arc (x,y) dans le choix des fourmis

+ la longueur de l'arc augmente, + p(x,y) diminue

$$p(x,y) = (\tau_{xy}^{\alpha}).(\eta_{xy}^{\beta}) + \gamma$$

+ il y a de phéromones, + p(x,y) augmente

les quantités de phéromones evoluent dans le temps :

évaporation : coefficient ρ

$$\tau_{xy}(t+1) = \tau_{xy}(t).(1-\rho) + Q$$

Et Q dépendant de $COST_{path}$

augmentation : coefficient Q

Méthodes limitant l'obtention d'un minimum local :

Avec $P_{MIN} \leq \tau_{xy}(t) \leq P_{MAX}$

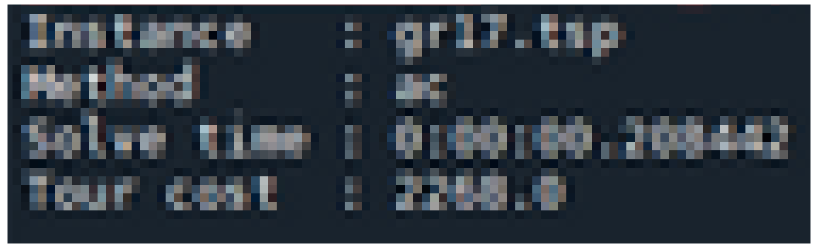
- fixer des quantités maximales et minimales de phéromones par arc
- considérer la qualité du chemin global dans l'ajout de phéromones
- valorisation de la recherche de nouveaux chemins

Ant colony algorithm

Principal inconvénient de cette méthode :

- Nombre trop important de variables : difficulté d'ajustement
- Grande variabilité des résultats observés
- Risque assez important d'obtenir un minimum local

```
Instance    : gr17.tsp
Method      : ac
Solve time  : 0:00:00.205272
Tour cost   : 2149.0
```



```
Instance    : gr17.tsp
Method      : ac
Solve time  : 0:00:00.208442
Tour cost   : 2268.0
```

Complexité de l'algorithme :

- Critère d'arrêt : stagnation du meilleur coût OU nombre d'itérations < 10.n
- Recherche d'un chemin pour chaque fourmi : n itérations
- Mise à jour de la matrice des poids : n^2 itérations

$$\Theta(n^3)$$

Nearest Neighbor algorithm

Principe de l'algorithme :

⟶ On choisit une ville de départ

- On cherche le plus proche voisin de cette ville, qu'on note "visité" et on stocke la distance correspondante ;
- On cherche ensuite le plus proche voisin "non-visité" de la dernière ville visitée;
- On répète cette opération n-1 fois;
- On rajoute la distance reliant la dernière ville visitée à la ville de départ.

⟶ On refait la même chose en démarrant dans chaque ville

⟶ On retient la trajectoire la moins couteuse

Nearest Neighbor algorithm

Complexité en temps : $\Theta(n^3)$ ou $\Theta(n^2)$

Résultats variables, relativement proches de la solution

```
Instance   : gr17.tsp
Method     : nn
Solve time : 0:00:00.004987
Tour cost  : 2178.0
```

```
Instance   : gr21.tsp
Method     : nn
Solve time : 0:00:00.006116
Tour cost  : 2891.0
```

```
Instance   : gr24.tsp
Method     : nn
Solve time : 0:00:00.008976
Tour cost  : 1526.0
```

Méthodes exactes

rapide, même sur des problèmes
de grande dimension

lent sur des problèmes
assez grands (complexité
spatiale)

assez lent : complexités en temps
et en espace importants

DFJ

MTZ

Held Karp

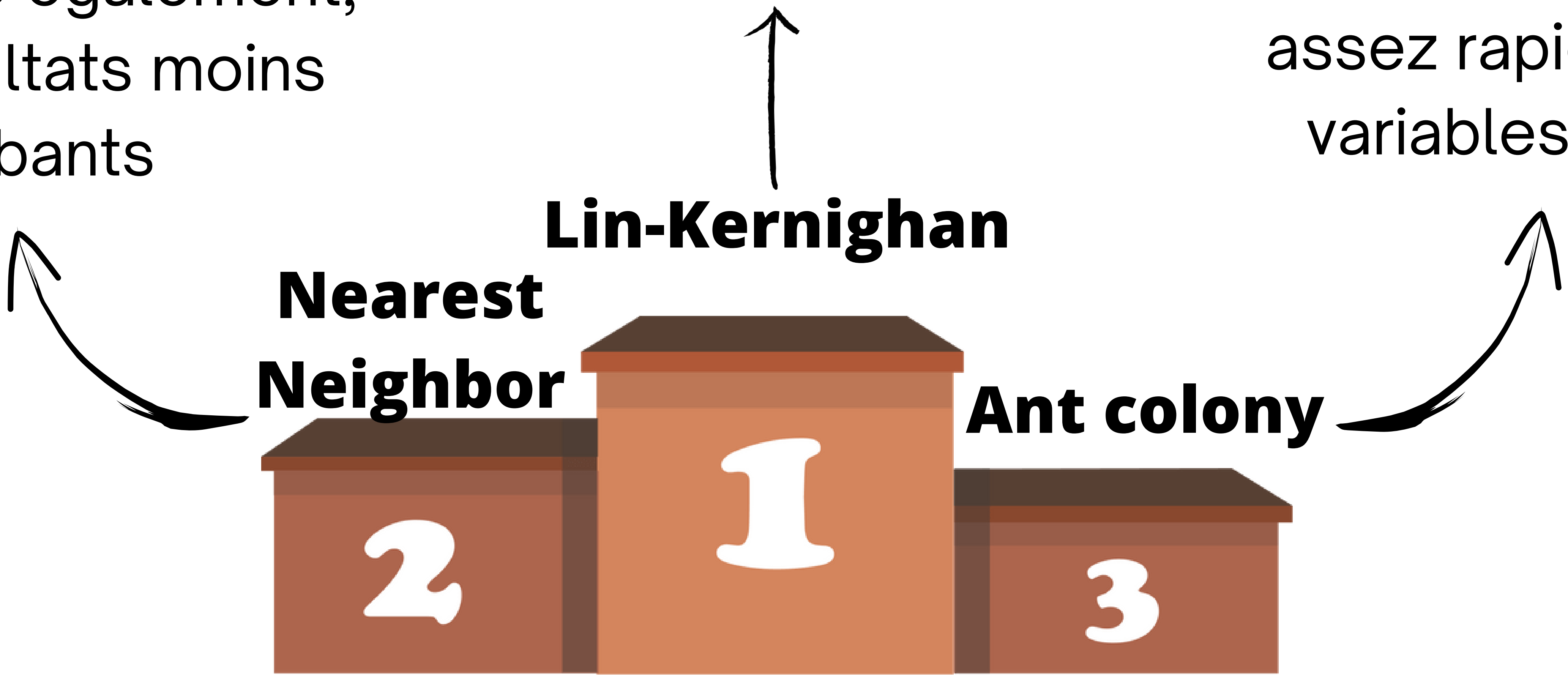
Raison majeure : utilisation d'un solveur utilisant du langage C ou C++

Méthodes heuristiques :

très rapide, et donne une solution convenable

très rapide également, mais résultats moins probants

assez rapide, mais résultats très variables (donc moins fiables)



Compromis à trouver entre rapidité et qualité de l'approximation

Merci pour votre écoute

→