Algorithmique et Complexité
Cours 7/7 : Approche des problèmes difficiles

CentraleSupélec – Gif

ST2 – Gif

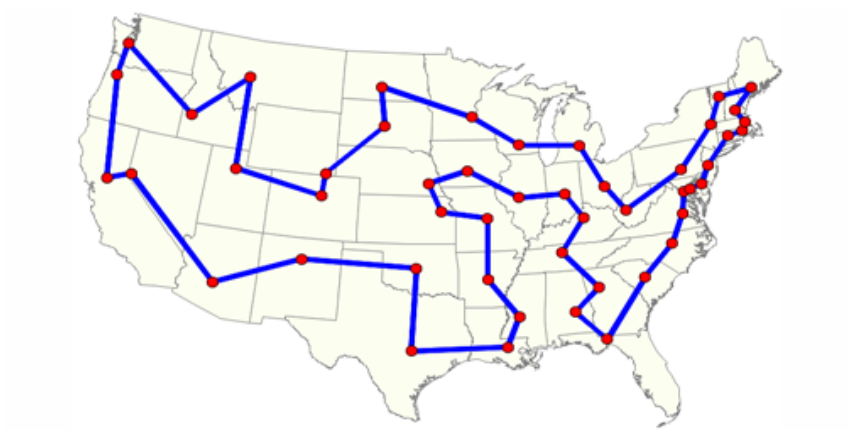
Plan

1. Problème du Voyageur de Commerce
   - Formalisation du problème
   - Complexité
2. Méthodes exactes
3. Heuristiques et approximation
4. Conclusion

Problématique rencontrée

Problème concret

- Étant donné un ensemble de villes et les distances entre ces villes, quel est le plus court chemin qui passe une fois par chaque ville et termine à la ville de départ ?



C'est le problème du voyageur de commerce (en anglais *Traveling Salesman Problem* ou TSP).

Voyageur de commerce (TSP)

Problème d'optimisation

- Instance :
  - $G = (V, E)$ un graphe complet et non orienté avec $|V| = n$
  - $d : E \to \mathbb{R}$ une fonction de poids qui associe une distance à chaque arête

Voyageur de commerce (TSP)

Problème d'optimisation

- Instance :
  - $G = (V, E)$ un graphe complet et non orienté avec $|V| = n$
  - $d : E \to \mathbb{R}$ une fonction de poids qui associe une distance à chaque arête
- Question :
  - Trouver $S = [s_1, ..., s_n]$ une liste d'éléments de V, tel que :

Voyageur de commerce (TSP)

Problème d'optimisation

- Instance :
  - $G = (V, E)$ un graphe complet et non orienté avec $|V| = n$
  - $d : E \to \mathbb{R}$ une fonction de poids qui associe une distance à chaque arête
- Question :
  - Trouver $S = [s_1, ..., s_n]$ une liste d'éléments de V, tel que :
- Contraintes :
  - Chaque élément de V apparaît exactement une fois dans S
  - On **minimise** $Score(S) = \sum_{s_i \in S} d(s_i, s_{i+1})$

(on pose $s_{n+1} = s_1$ pour simplifier les notations)

Comment résoudre ce problème ?

1. Parcourir les bibliothèques de problèmes connus afin de connaitre les résultats existants
   - exemple : le *compendium* de Viggo Kann.

Comment résoudre ce problème ?

1. Parcourir les bibliothèques de problèmes connus afin de connaitre les résultats existants
   - exemple : le *compendium* de Viggo Kann.

2. Imaginons (ce qui est faux) que ce problème ne se trouve pas dans la littérature, nous allons devoir l'étudier en commençant par cette question :
   ➜ Est-il dans *NP* ?

TSP comme problème de décision

Problème de **décision**

- Instance :
  - $G = (V, E)$ un graphe complet non orienté avec $|V| = n$
  - $d : E \to \mathbb{R}$ une fonction de poids qui associe une distance à chaque arête
  - $B \in \mathbb{R}$ une borne max
- Question :
  - Existe-t-il $S = [s_1, ..., s_n]$ une liste d'éléments de V ?
- Contraintes :
  - Chaque élément de V apparaît exactement une fois dans S
  - $\sum_{s_i \in S} d(s_i, s_{i+1}) \leq B$

TSP comme problème de décision

TSP est dans *NP* ?

L'algorithme vérifiant une solution S d'une instance positive (V, E) du problème doit vérifier les deux contraintes du problème :

- Chaque élément de V apparaît exactement une fois dans S
- $\sum_{s_i \in S} d(s_i, s_{i+1}) \leq B$

TSP comme problème de décision

TSP est dans *NP* ?

L'algorithme vérifiant une solution S d'une instance positive (V, E) du problème doit vérifier les deux contraintes du problème :

- Chaque élément de V apparaît exactement une fois dans S
  - ➜ réalisable en $\mathcal{O}(n)$ $(n = |V|)$
- $\sum_{s_i \in S} d(s_i, s_{i+1}) \leq B$
  - ➜ réalisable en $\mathcal{O}(n)$ (avec une matrice d'adjacence)

TSP comme problème de décision

TSP est dans *NP* ?

L'algorithme vérifiant une solution S d'une instance positive (V, E) du problème doit vérifier les deux contraintes du problème :

- Chaque élément de V apparaît exactement une fois dans S
  - ➜ réalisable en $\mathcal{O}(n)$ $(n = |V|)$
- $\sum_{s_i \in S} d(s_i, s_{i+1}) \leq B$
  - ➜ réalisable en $\mathcal{O}(n)$ (avec une matrice d'adjacence)

On a donc un algorithme polynomial pour vérifier une solution.

Conclusion

➜ Le problème du voyageur de commerce est bien dans *NP*.

TSP comme problème de décision

TSP est-il *NP*-complet ?

Maintenant que l'on sait que le problème est dans *NP*, il va falloir :

- Soit trouver un algorithme de résolution polynomial ;
- Soit montrer qu'il est *NP*-complet.

TSP comme problème de décision

TSP est-il *NP*-complet ?

Maintenant que l'on sait que le problème est dans *NP*, il va falloir :

- Soit trouver un algorithme de résolution polynomial ;
- Soit **montrer qu'il est *NP*-complet**.
  . . . en effectuant une réduction polynomiale depuis un problème connu

Liste de problèmes déjà rencontrés

- Clique
- Stable
- **HAM/D-HAM**
- SAT
- ...

Cycle hamiltonien

Problème du cycle hamiltonien HAM

Instance :

- $G = (V, E)$ un graphe non orienté avec $|V| = n$

Question :

- Existe-t-il $S = [s_1, ..., s_n]$ une liste ordonnée d'éléments de V, tel que

Contraintes :

- Chaque élément de V apparaît exactement une fois dans S
- $\forall s_i \in S, \{s_i, s_{i+1}\} \in E$

Cycle hamiltonien

Problème du cycle hamiltonien HAM

Instance :

- $G = (V, E)$ un graphe non orienté avec $|V| = n$

Question :

- Existe-t-il $S = [s_1, ..., s_n]$ une liste ordonnée d'éléments de V, tel que

Contraintes :

- Chaque élément de V apparaît exactement une fois dans S
- $\forall s_i \in S, \{s_i, s_{i+1}\} \in E$

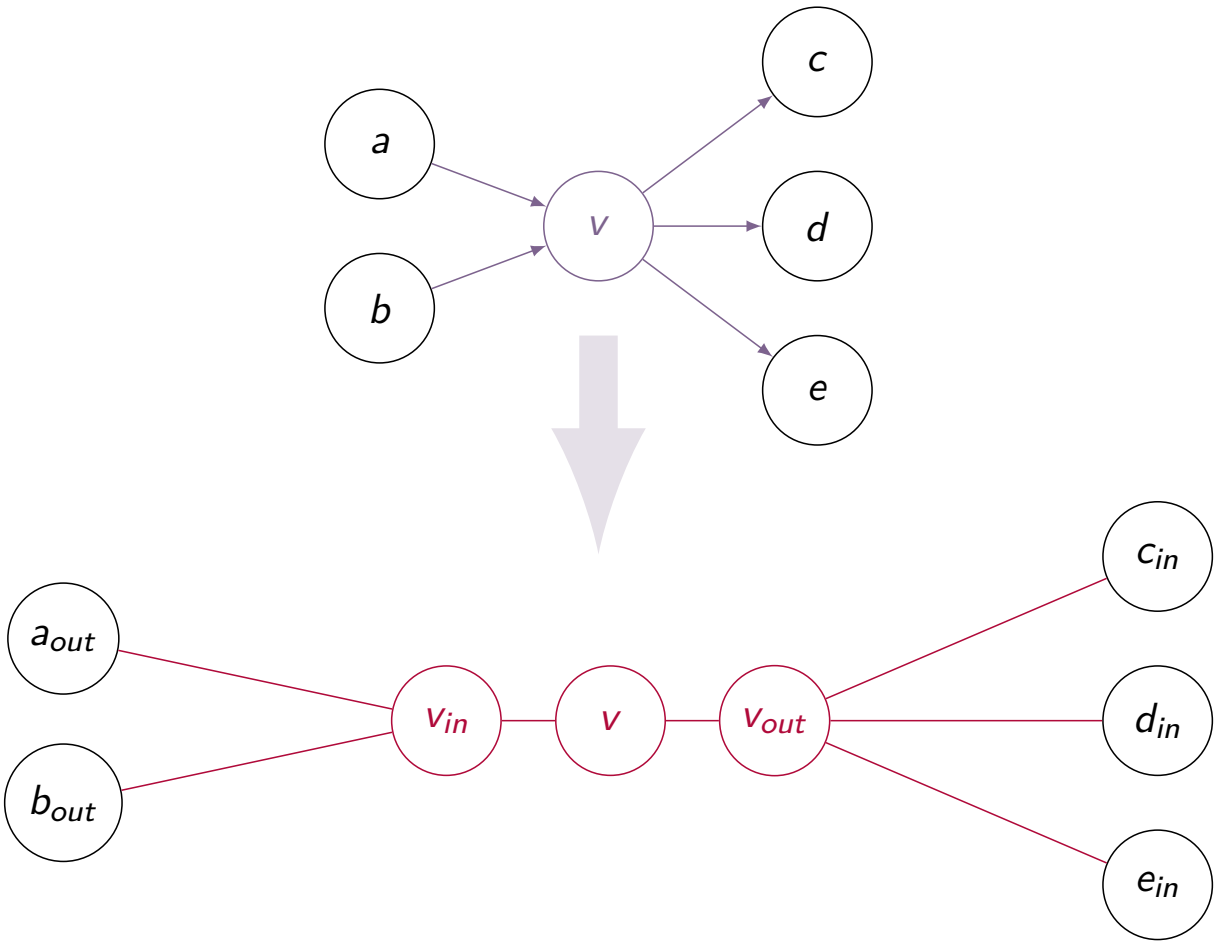
À savoir : HAM est NP-complet

D-HAM vu au cours précédent se réduit en HAM

⏩ skip reduction

D-HAM $\leq$ HAM

- Une réduction courante pour passer d'un problème de graphe orienté vers un problème de graphe non-orienté :



Réduction polynomiale : HAM $\leq$ TSP

Principe

On cherche une réduction polynomiale de chaque instance du cycle hamiltonien en une instance du voyageur de commerce

Si c'est le cas, alors HAM $\leq$ TSP et comme HAM est NP-difficile et TSP est NP, alors TSP est NP-complet...

Réduction polynomiale : HAM $\leq$ TSP

Principe

On cherche une réduction polynomiale de chaque instance du cycle hamiltonien en une instance du voyageur de commerce

Si c'est le cas, alors HAM $\leq$ TSP et comme HAM est NP-difficile et TSP est NP, alors TSP est NP-complet...

Réduction

Étant donnée une instance de HAM $G = (V, E)$,

Réduction polynomiale : HAM $\leq$ TSP

Principe

On cherche une réduction polynomiale de chaque instance du cycle hamiltonien en une instance du voyageur de commerce

Si c'est le cas, alors HAM $\leq$ TSP et comme HAM est NP-difficile et TSP est NP, alors TSP est NP-complet...

Réduction

Étant donnée une instance de HAM $G = (V, E)$,

on construit une instance de TSP $\langle G' = (V', E'), d, B \rangle$

Réduction polynomiale : HAM $\leq$ TSP

Principe

On cherche une réduction polynomiale de chaque instance du cycle hamiltonien en une instance du voyageur de commerce

Si c'est le cas, alors HAM $\leq$ TSP et comme HAM est NP-difficile et TSP est NP, alors TSP est NP-complet...

Réduction

Étant donnée une instance de HAM $G = (V, E)$,

on construit une instance de TSP $\langle G' = (V', E'), d, B \rangle$

telle que pour toute instance de HAM, s'il existe une solution, alors il existe aussi une solution dans l'instance de TSP construite, et réciproquement...

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' =$
- $E' =$

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' = V$
  On garde les mêmes sommets...
- $E' =$

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' = V$
  On garde les mêmes sommets...
- $E' = \{\{u, v\}, \forall u, v \in V \wedge u \neq v\}$
  ... mais on construit un graphe G' complet !

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' = V$
  On garde les mêmes sommets...
- $E' = \{\{u, v\}, \forall u, v \in V \wedge u \neq v\}$
  ... mais on construit un graphe G' complet !
- $\forall e \in E'$,
  $d(e) =$
- $B =$

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' = V$
  On garde les mêmes sommets...
- $E' = \{\{u, v\}, \forall u, v \in V \wedge u \neq v\}$
  ... mais on construit un graphe G' complet !
- $\forall e \in E'$,
  $d(e) = 0$ if $e \in E$,
  $d(e) = 1$ if $e \notin E$
- $B =$

Réduction Polynomiale

Réduction

Soit $G = (V, E)$ une instance de HAM, on construit $\langle G' = (V', E'), d, B \rangle$ avec :

- $V' = V$
  On garde les mêmes sommets...
- $E' = \{\{u, v\}, \forall u, v \in V \wedge u \neq v\}$
  ... mais on construit un graphe G' complet !
- $\forall e \in E'$,
  $d(e) = 0$ if $e \in E$,
  $d(e) = 1$ if $e \notin E$
- $B = 0$

Réduction Polynomiale 2/2

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

Réduction Polynomiale 2/2

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM

Réduction Polynomiale

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM
S définit aussi un cycle dans G' de poids 0 *(tous les $d(e)$ valent 0)*

➜ Donc il existe une solution pour TSP.

Réduction Polynomiale 2/2

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM
S définit aussi un cycle dans G' de poids 0 *(tous les $d(e)$ valent 0)*

➜ Donc il existe une solution pour TSP.

$\Longleftarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance $\langle G', d, B \rangle$ de TSP *obtenue par transformation depuis G*

Réduction Polynomiale

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM
S définit aussi un cycle dans G' de poids 0 *(tous les $d(e)$ valent 0)*

➜ Donc il existe une solution pour TSP.

$\Longleftarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance $\langle G', d, B \rangle$ de TSP *obtenue par transformation depuis G*

$B = 0$ et $d(e) = 1$ pour tout $e \notin E$, donc la solution emprunte uniquement des arêtes de E !

Réduction Polynomiale 2/2

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM
S définit aussi un cycle dans G' de poids 0 *(tous les $d(e)$ valent 0)*

➜ Donc il existe une solution pour TSP.

$\Longleftarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance $\langle G', d, B \rangle$ de TSP *obtenue par transformation depuis G*

$B = 0$ et $d(e) = 1$ pour tout $e \notin E$, donc la solution emprunte uniquement des arêtes de E !

➜ S définit aussi une solution à l'instance G de HAM

Par contraposition : pas de sol. à l'instance HAM $\Rightarrow$ pas de sol. pour TSP

Réduction Polynomiale 2/2

Montrons que c'est bien une réduction polynomiale

NB : *La transformation est polynomiale (vérifier chaque opération)*

$\Longrightarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance G de HAM
S définit aussi un cycle dans G' de poids 0 *(tous les $d(e)$ valent 0)*

➜ Donc il existe une solution pour TSP.

$\Longleftarrow$ Soit $S = [s_1, ..., s_n]$ une solution de l'instance $\langle G', d, B \rangle$ de TSP *obtenue par transformation depuis G*

$B = 0$ et $d(e) = 1$ pour tout $e \notin E$, donc la solution emprunte uniquement des arêtes de E !

➜ S définit aussi une solution à l'instance G de HAM

Par contraposition : pas de sol. à l'instance HAM $\Rightarrow$ pas de sol. pour TSP

$$\text{HAM} \leq \text{TSP}$$

Complexité TSP

TSP est *NP*-Complet

- Voyageur de commerce est dans *NP*
- Cycle hamiltonien est *NP*-complet
- Cycle hamiltonien $\leq$ Voyageur de commerce

Complexité TSP

TSP est *NP*-Complet

- Voyageur de commerce est dans *NP*
- Cycle hamiltonien est *NP*-complet
- Cycle hamiltonien $\leq$ Voyageur de commerce

➜ Voyageur de commerce (comme problème de décision) est *NP*-complet

Complexité TSP

TSP est *NP*-Complet

- Voyageur de commerce est dans *NP*
- Cycle hamiltonien est *NP*-complet
- Cycle hamiltonien $\leq$ Voyageur de commerce

➜ Voyageur de commerce (comme problème de décision) est *NP*-complet

Conclusion (problème d'optimisation)

Il n'est pas possible de calculer une solution optimale en temps polynomial (sauf si P=NP ...)

Résoudre des problèmes d'optimisation NP-difficiles

Méthodes exactes

On recherche la meilleure solution... en essayant d'être efficace !

Exemples : Backtracking, Branch & Bound, Programmation linéaire, ...

Résoudre des problèmes d'optimisation NP-difficiles

Méthodes exactes

On recherche la meilleure solution... en essayant d'être efficace !

Exemples : Backtracking, Branch & Bound, Programmation linéaire, ...

Méthodes pas nécessairement exactes, mais en temps polynomial

- Algorithmes heuristiques et algorithmes d'approximation
  ➜ ex : gloutons, voir plus loin dans le cours
- Algorithmes randomisés (Monte Carlo, Las Vegas)
- Méthodes générales d'exploration de l'espace des solutions
  ➜ métaheuristiques (ex : recuit simulé, algorithmes génétiques ...)

Plan

1 Problème du Voyageur de Commerce

2 Méthodes exactes
- Force Brute
- Backtracking
- Espace des solutions
- Algorithme
- Amélioration

3 Heuristiques et approximation

4 Conclusion

Résoudre des problèmes d'optimisation NP-difficiles

Méthodes exactes

On recherche la meilleure solution... en essayant d'être efficace !

Exemples : **Force Brute**, **Backtracking**, Branch & Bound, Programmation linéaire, ...

Force brute (exploration exhaustive)

Principe

1. Énumérer successivement l'ensemble des configurations.
   Toutes les solutions possibles !
2. Évaluer le score de chaque configuration
3. Conserver la meilleure configuration

Force brute (exploration exhaustive)

Principe

1. Énumérer successivement l'ensemble des configurations.
   Toutes les solutions possibles !

   en TSP : toutes les listes de $|V|$ sommets (ensemble des cycles possibles)

2. Évaluer le score de chaque configuration

   en TSP : calculer pour chaque cycle la somme des poids de ses arêtes.

3. Conserver la meilleure configuration

   en TSP : choisir le cycle avec le score le plus faible.

Force brute (exploration exhaustive)

Principe

1. Énumérer successivement l'ensemble des configurations.
   Toutes les solutions possibles !

   en TSP : toutes les listes de $|V|$ sommets (ensemble des cycles possibles)

   ➜ $\frac{|V-1|!}{2}$ solutions possibles

2. Évaluer le score de chaque configuration

   en TSP : calculer pour chaque cycle la somme des poids de ses arêtes.

3. Conserver la meilleure configuration

   en TSP : choisir le cycle avec le score le plus faible.

Complexité exponentielle

Backtracking

Principe

- Construction itérative des solutions

➜ Étudier l'ensemble des configurations possibles

Backtracking

Principe

- Construction itérative des solutions

➜ Étudier l'ensemble des configurations possibles

Exemple :

En TSP, à chaque étape, on peut séparer les cas selon :

- Option 1 : le prochain noeud à visiter
- Option 2 : garder ou éliminer une arête

Backtracking

Principe

- Construction itérative des solutions

➜ Étudier l'ensemble des configurations possibles

Exemple :

En TSP, à chaque étape, on peut séparer les cas selon :

- Option 1 : le prochain noeud à visiter
- Option 2 : garder ou éliminer une arête

- On parle d'**explorer** l'espace des solutions.

Backtracking

Principe

- Construction itérative des solutions

➜ Étudier l'ensemble des configurations possibles

Exemple :

En TSP, à chaque étape, on peut séparer les cas selon :

- Option 1 : le prochain noeud à visiter
- Option 2 : garder ou éliminer une arête

- On parle d'**explorer** l'espace des solutions.

➜ Le backtracking est une exploration par **séparation** de l'espace des solutions

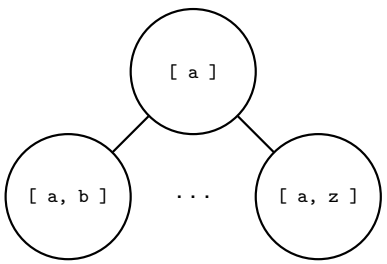
Parcours de l'espace des solutions

Principe

➜ Espace des solutions vu comme un arbre

Les branches représentent la construction itérative des solutions

Exemple : dans TSP, ajout d'un élément en fin de liste (option 1)



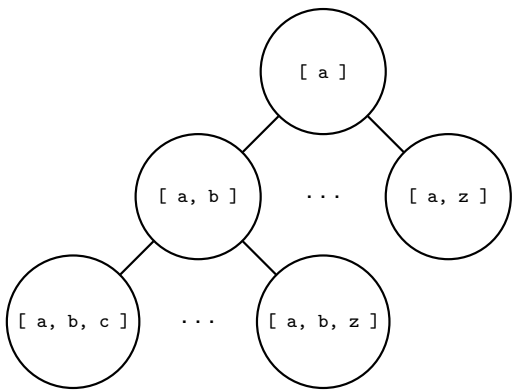
Parcours de l'espace des solutions

Principe

➜ Espace des solutions vu comme un arbre

Les branches représentent la construction itérative des solutions

Exemple : dans TSP, ajout d'un élément en fin de liste (option 1)



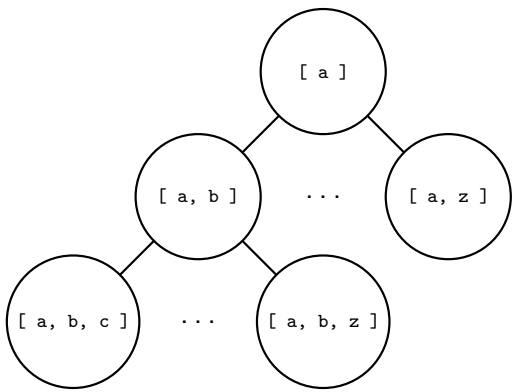
Parcours de l'espace des solutions

Principe

➜ Espace des solutions vu comme un arbre

Les branches représentent la construction itérative des solutions

Exemple : dans TSP, ajout d'un élément en fin de liste (option 1)

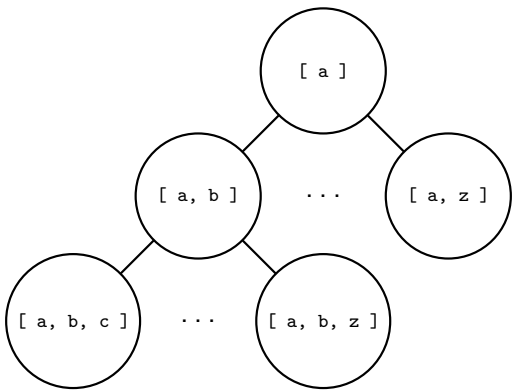


➜ Les feuilles contiennent les solutions possibles.

Parcours de l'espace des solutions

Principe

Exemple : dans TSP, ajout d'un élément en fin de liste (option 1)



➜ Espace des solutions vu comme un arbre

- Énumération des solutions par exploration en profondeur de cet arbre.

  → Le but est de choisir une solution optimale en examinant des solutions possibles dans les feuilles.

  → On parle d'arbre implicite, car il est construit au fur et au mesure de l'exploration. On ne représente pas l'arbre des solutions en mémoire !

Exemple du voyageur de commerce (option 2)

Séparation sur le choix d'une arête du cycle

- À chaque étape, on sépare l'ensemble des cycles hamiltoniens, entre ceux qui vont emprunter une arête choisie $\{i, j\}$ et ceux qui ne vont pas l'emprunter.

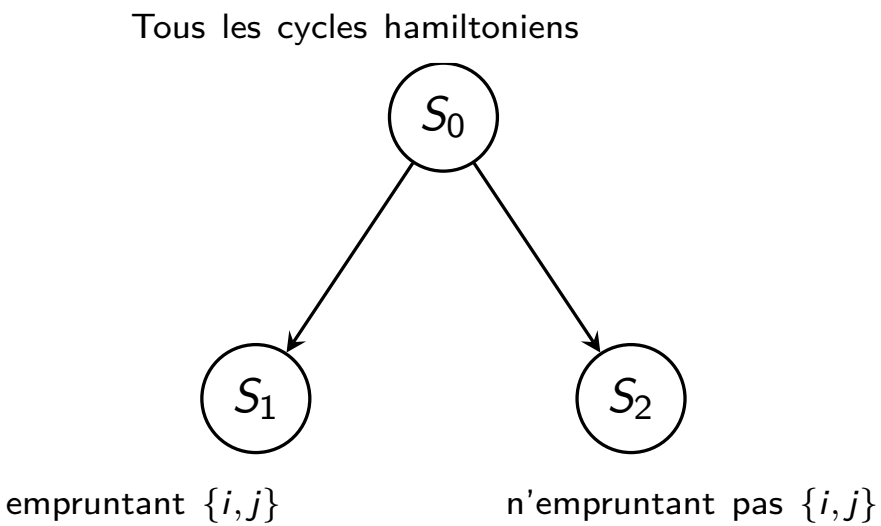
➜ Arbre binaire de hauteur $|E|$

Exemple du voyageur de commerce (option 2)

Séparation sur le choix d'une arête du cycle

- À chaque étape, on sépare l'ensemble des cycles hamiltoniens, entre ceux qui vont emprunter une arête choisie $\{i, j\}$ et ceux qui ne vont pas l'emprunter.

➜ Arbre binaire de hauteur $|E|$

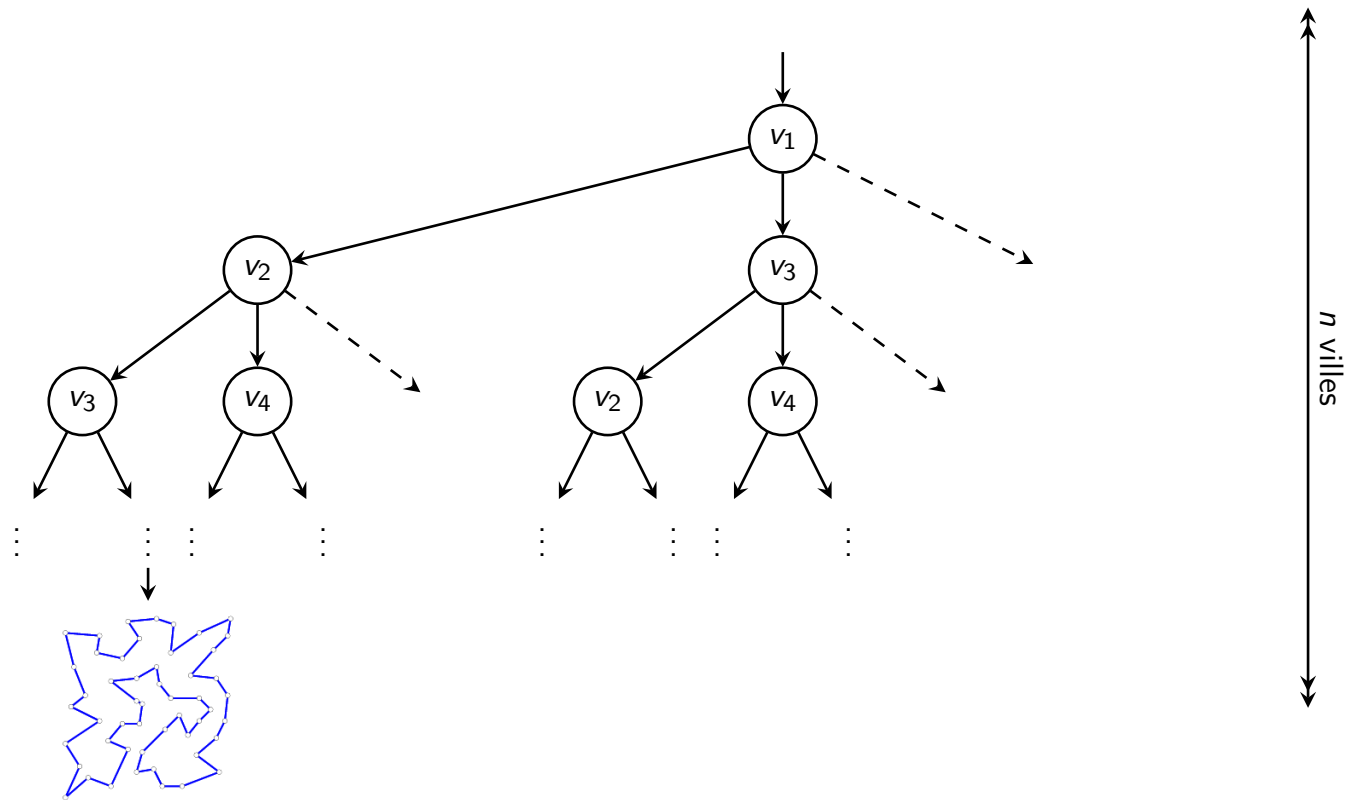


Exemple du voyageur de commerce (option 1)

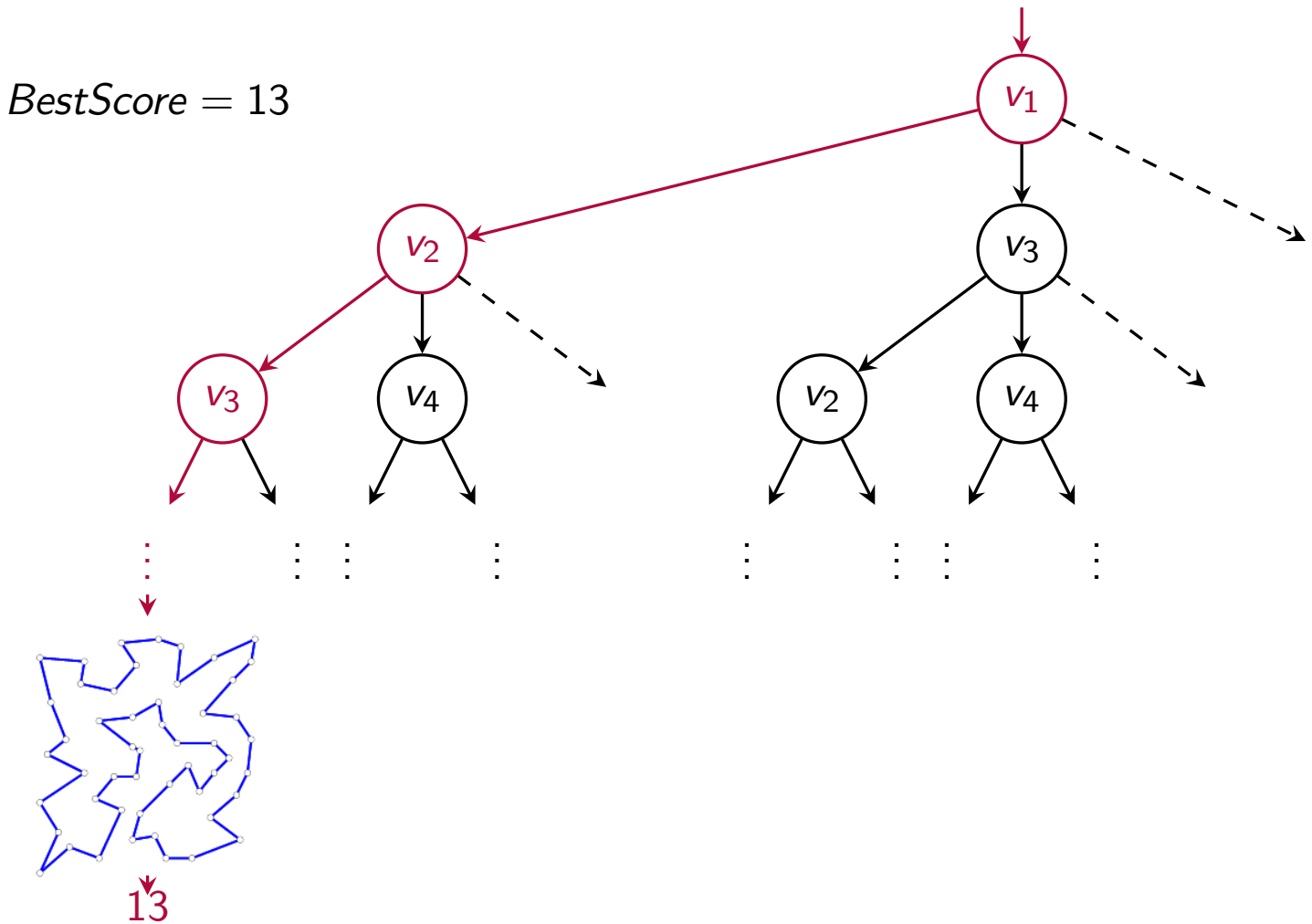
Séparation sur le choix du prochain noeud à visiter

- À chaque étape : choix d'une ville parmi les villes non encore visitées

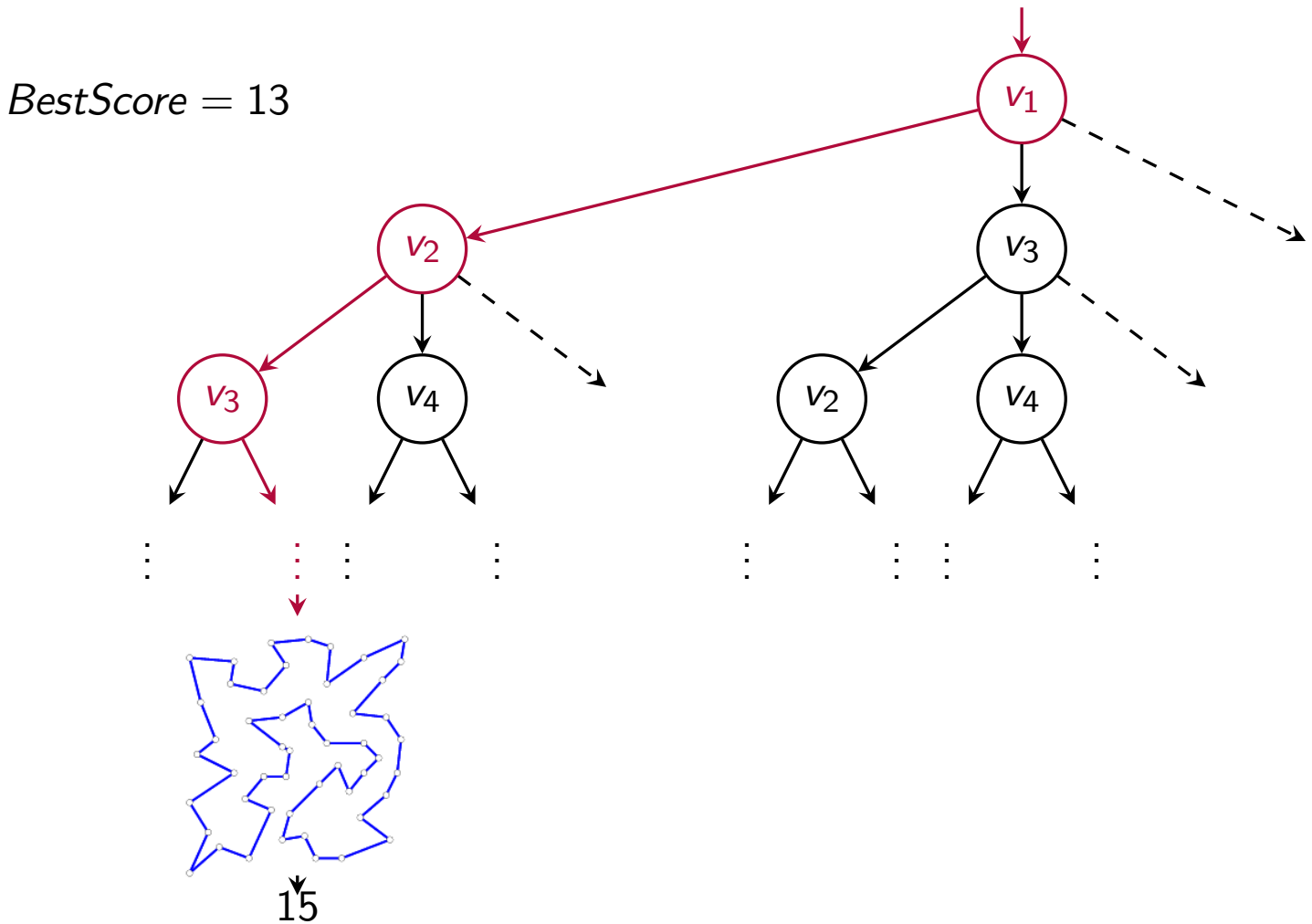
➜ Arbre : chaque nœud a autant de fils que de voisins



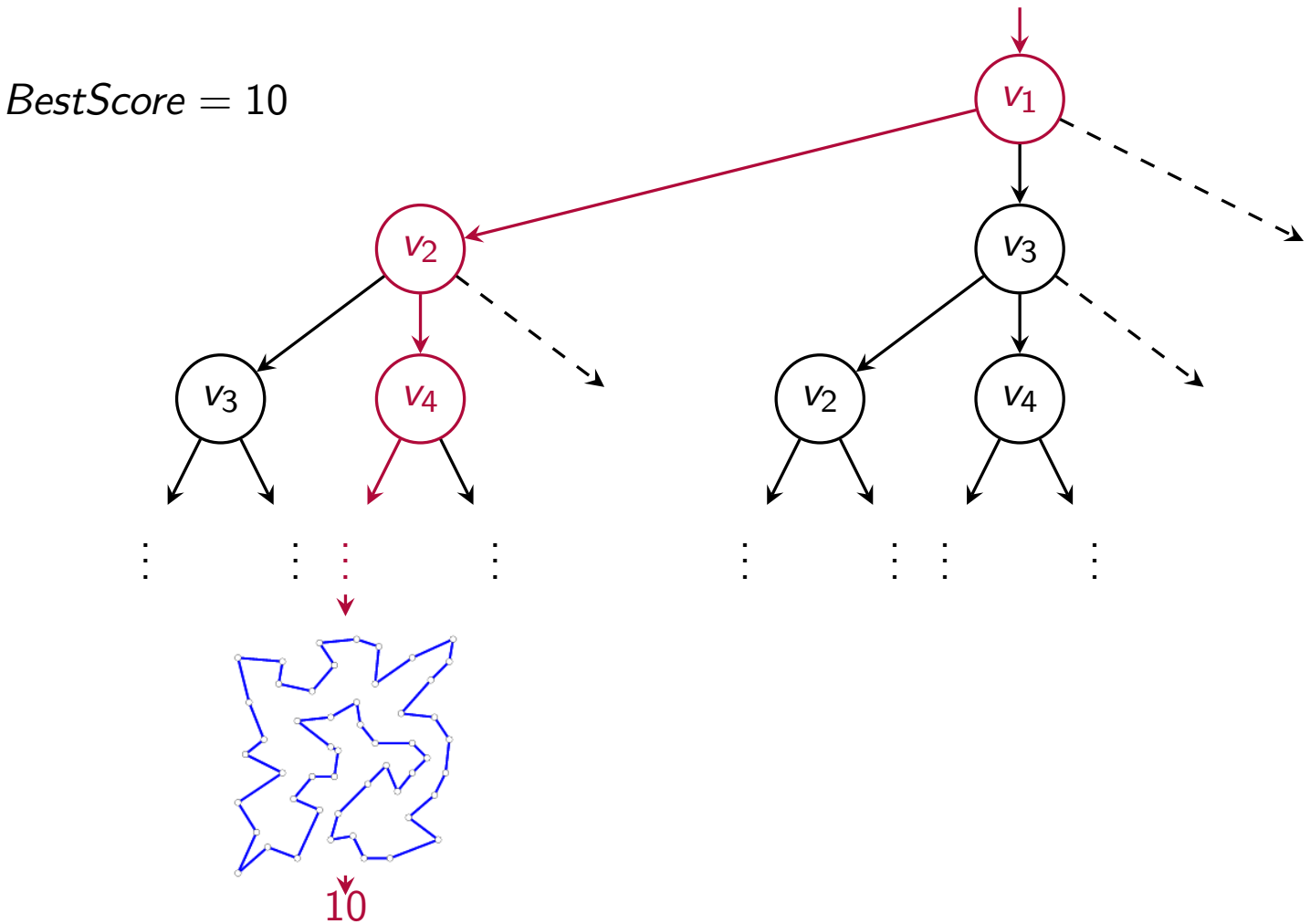
Algorithme de Backtracking



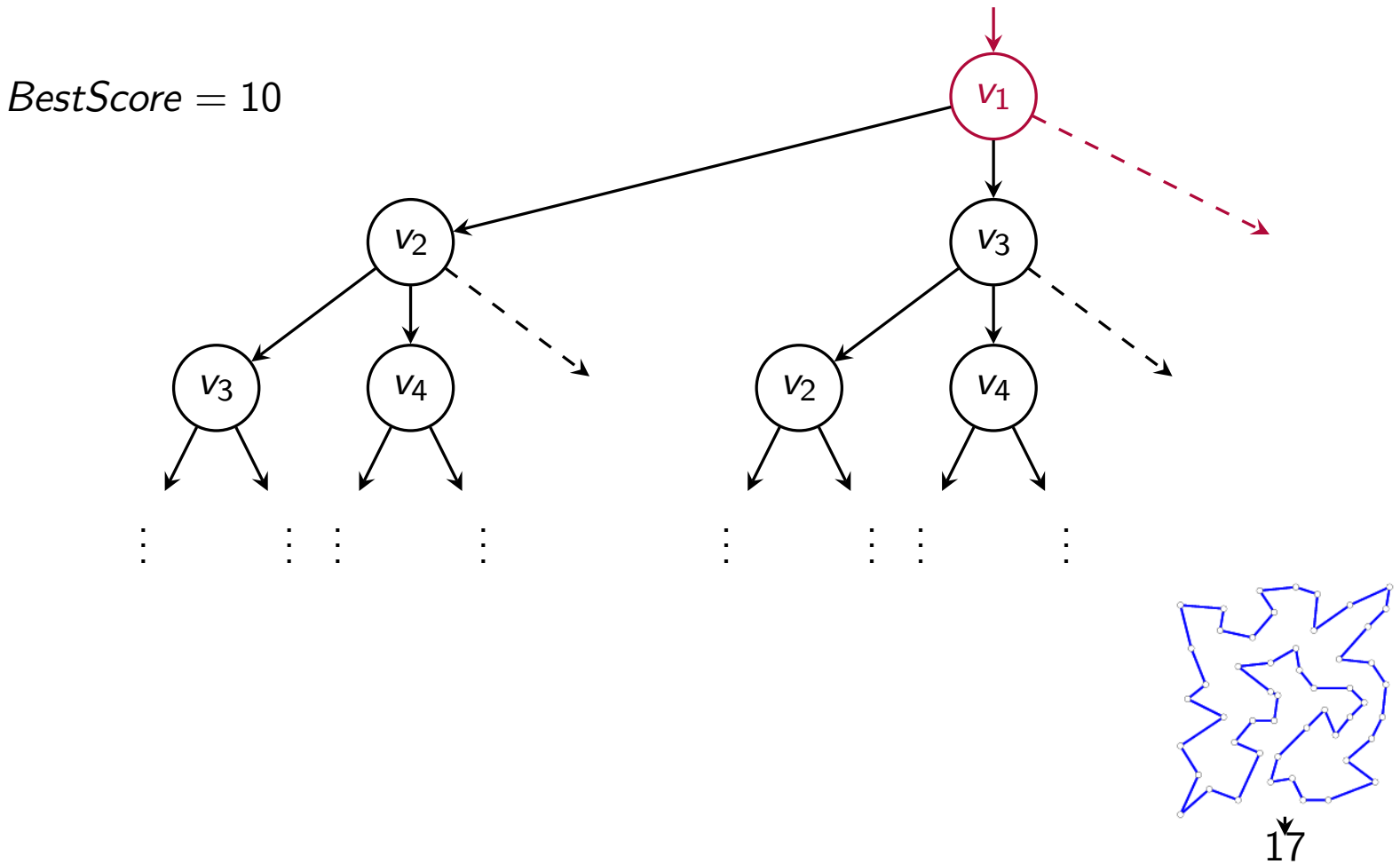
Algorithme de Backtracking



Algorithme de Backtracking



Algorithme de Backtracking



Algorithme de Backtracking

Principe général

Pour s une solution partielle ou réalisable, on suppose disposer des fonctions suivantes :

- $children(s)$: retourne les solutions atteignables en une étape depuis s
- $terminal(s)$: retourne *true* si la solution est réalisable, *false* sinon
- $score(s)$: retourne le score de la solution réalisable s

Algorithme de Backtracking

squelette de code

```
 1 bestScore = Inf
 2 bestSol = None
 3 def backtracking(s) :
 4     if terminal(s) :
 5         if score(s) < bestScore :
 6             bestScore = score(s)
 7             bestSol = s
 8     else :
 9         for c in children(s):
10             backtracking(c)
```

Améliorer l'algorithme

Comment améliorer cet algorithme ?

Améliorer l'algorithme

Comment améliorer cet algorithme ?

Principe

On explore en profondeur l'espace des solutions tout en coupant les branches non prometteuses. **Attention : des améliorations ne réduisent pas de la complexité qui restera exponentielle !**

Améliorer l'algorithme

Comment améliorer cet algorithme ?

Principe

On explore en profondeur l'espace des solutions tout en coupant les branches non prometteuses. **Attention : des améliorations ne réduisent pas de la complexité qui restera exponentielle !**

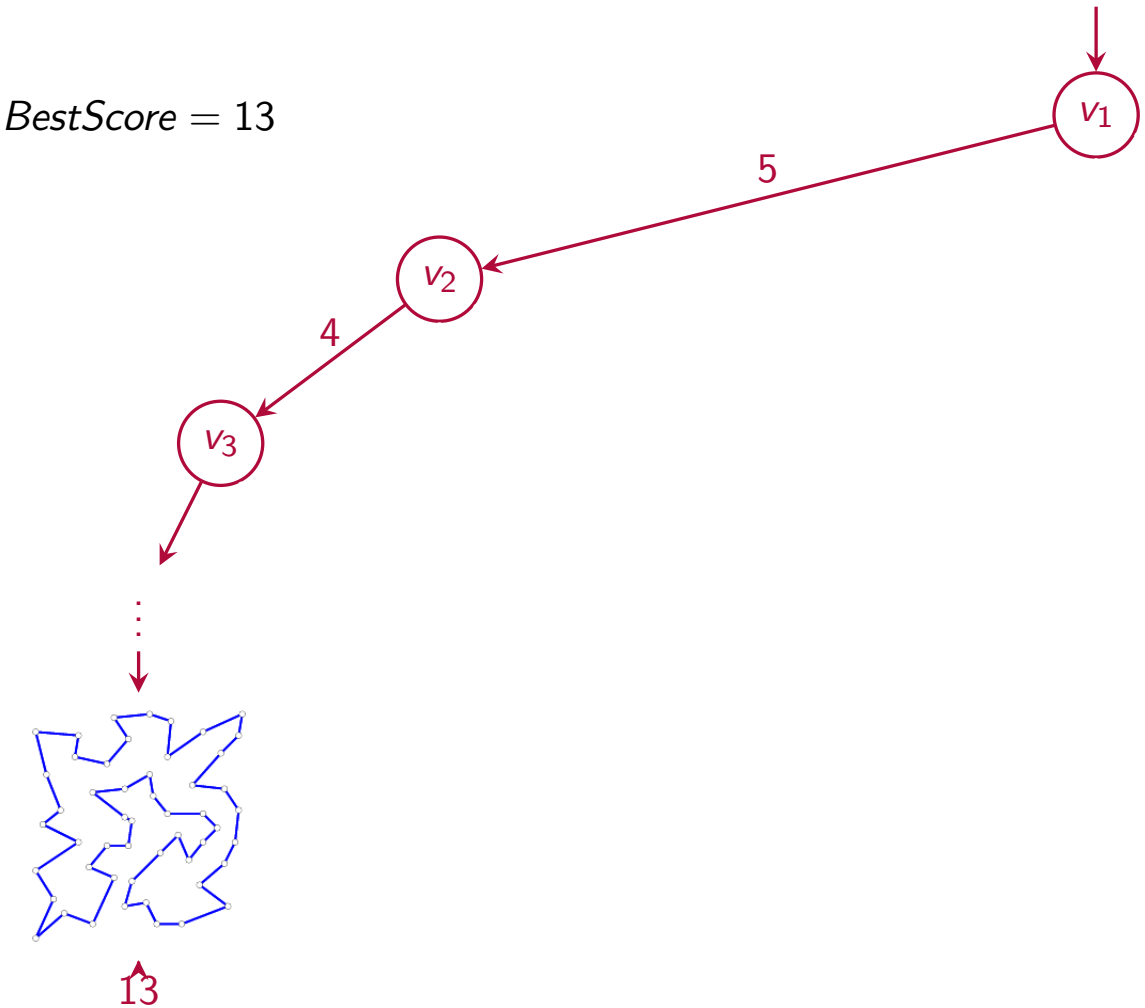
Cas du voyageur de commerce

- Noter le score de la meilleure solution réalisable trouvée : *BestScore*
- Une branche = une solution partielle $s = [s_1, ..., s_k]$
  (donc un début de chemin hamiltonien)

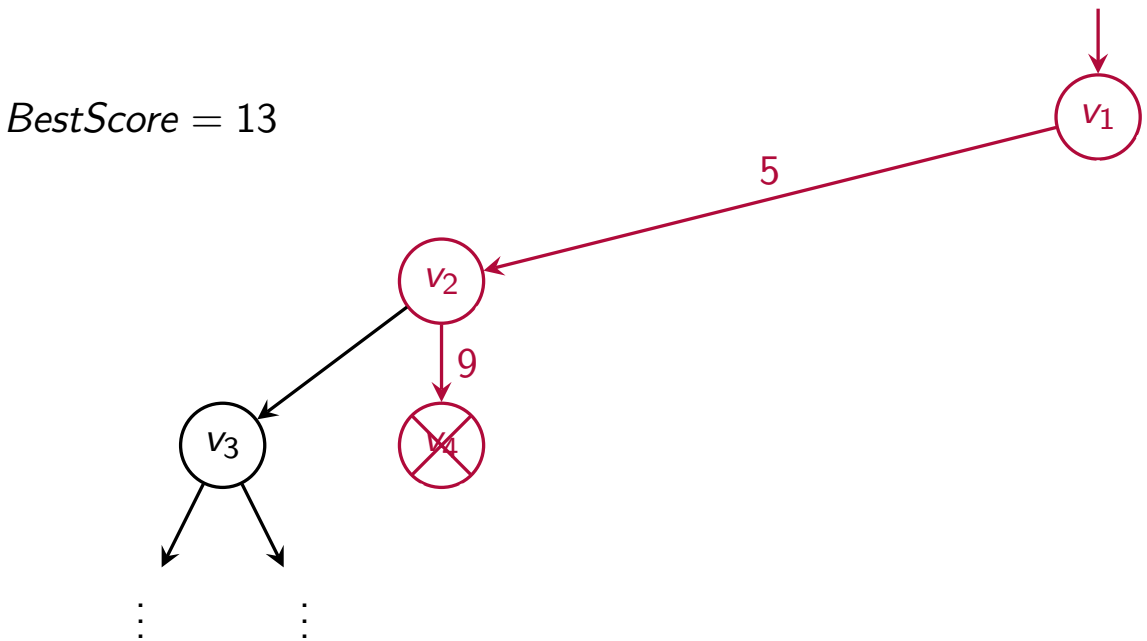
➜ Une branche non prometteuse = dont la solution partielle a déjà un score plus élevé que *BestScore* : $\sum_{s_i \in s} d(s_i, s_{i+1}) > BestScore$

✗ On abandonne l'exploration de cette branche !

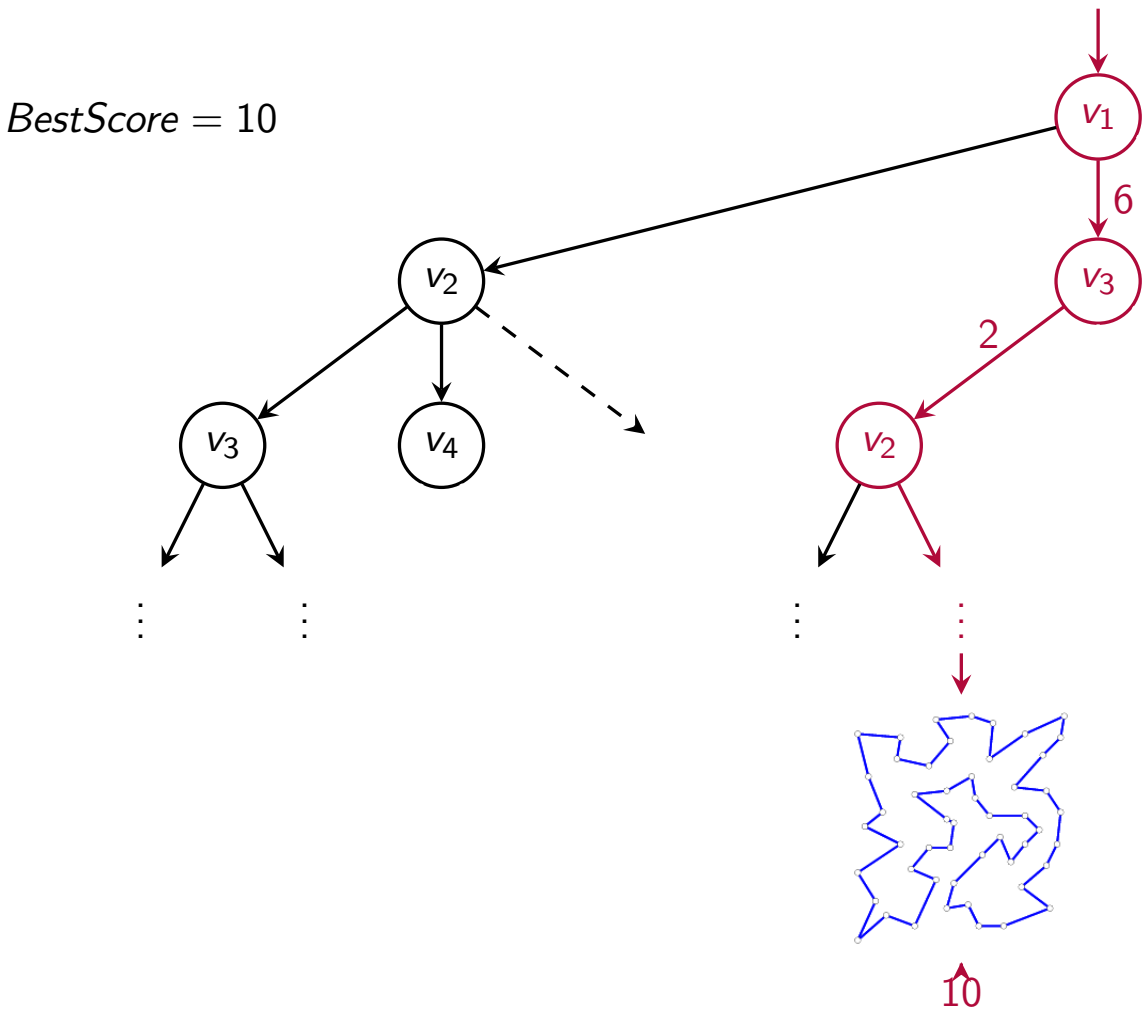
Algorithme de Backtracking : version améliorée



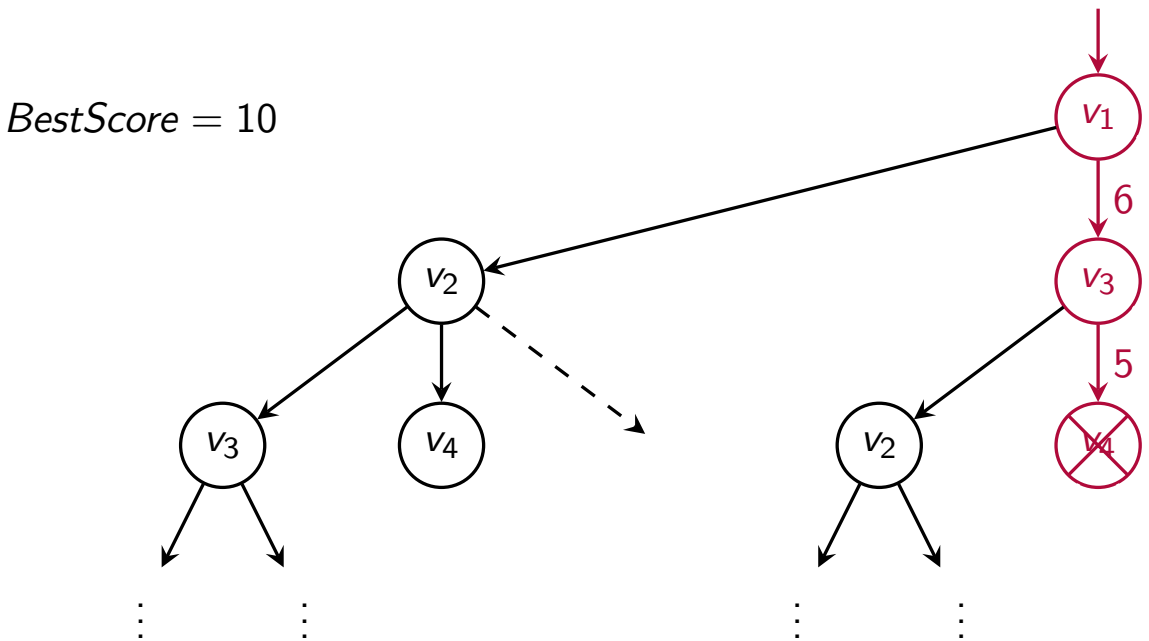
Algorithme de Backtracking : version améliorée



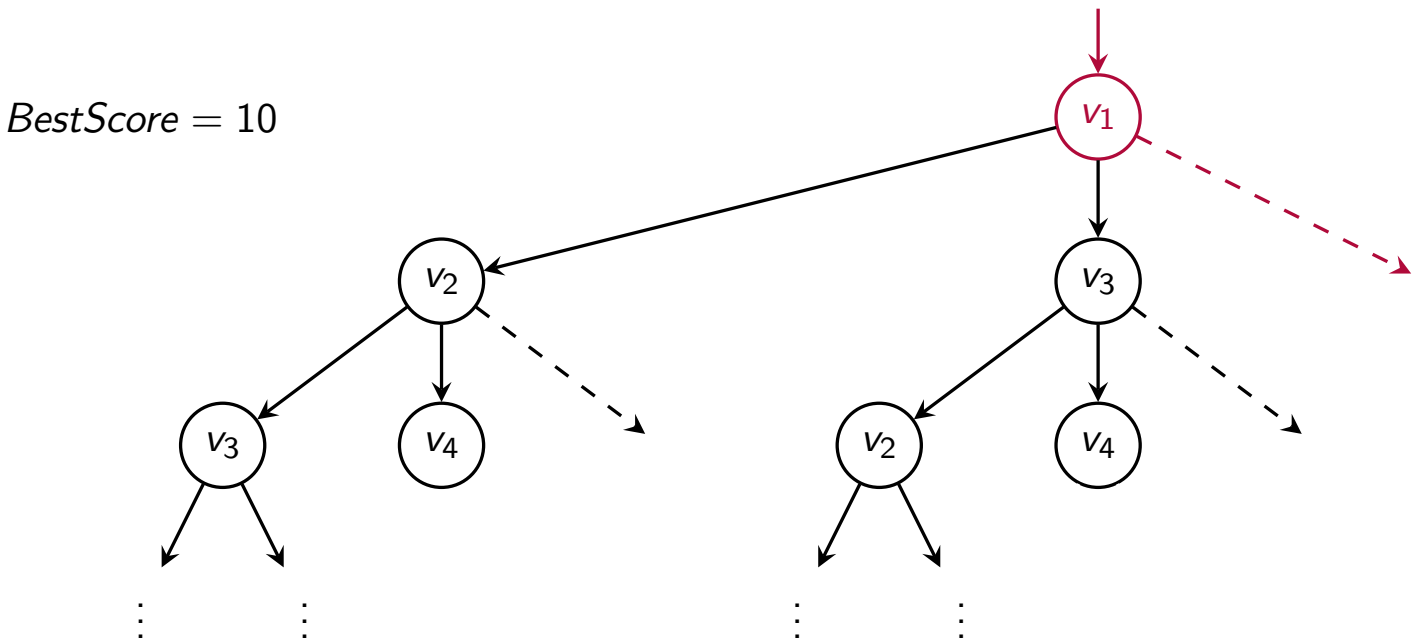
Algorithme de Backtracking : version améliorée



Algorithme de Backtracking : version améliorée



Algorithme de Backtracking : version améliorée



Plan

1 Problème du Voyageur de Commerce

2 Méthodes exactes

3 Heuristiques et approximation

4 Conclusion

Algorithmes en temps polynomial pour résoudre les problèmes difficiles

Problème

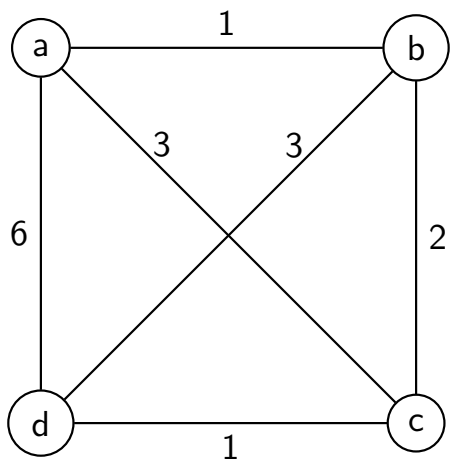
Les algorithmes produisant une solution optimale ont une complexité exponentielle :

- ils peuvent traiter des instances de petite taille

Pour des instances de grande taille, il faut se contenter d'une solution qui ne sera pas forcément optimale. . .

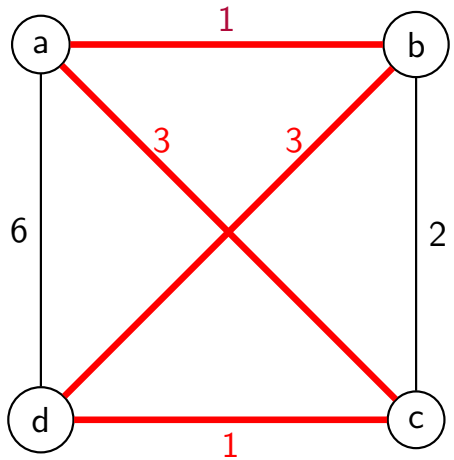
Prenons une instance TSP...

...d'une si petite taille que la solution optimale saute aux yeux (même nus)



Prenons une instance TSP...

...d'une si petite taille que la solution optimale saute aux yeux (même nus)



Solution optimale = 1+3+1+3=8

Greedy TSP

Idée d'un algorithme glouton

1. Choisir arbitrairement un sommet de départ v
2. Répeter jusqu'à ce que le tour complet soit fermé
   1. Choisir entre les voisins de v celui qui est le plus proche et ne fait pas déjà partie du tour, v',
   2. v' devient un nouveau sommet courant, $v \leftarrow v'$.

Et sa complexité

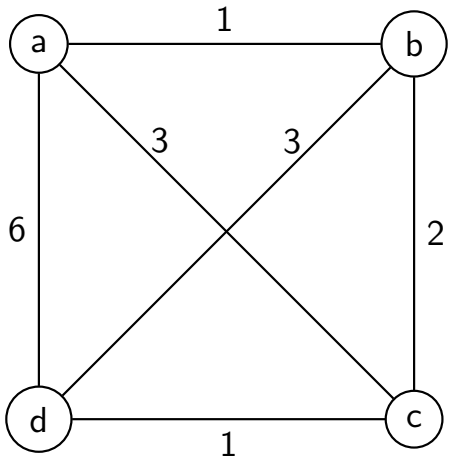
Notre algorithme est en temps polynomial (car il ressemble à un parcours de graphe)

⏩ skip greedy en python

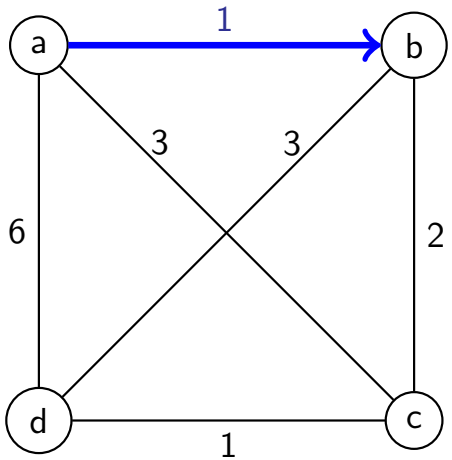
Une possible implémentation (basée sur DFS, le graphe vu comme une matrice)

```
 1 def ClstNeighbor_TSP(graphe,v): # v - courant
 2     tour.append(v)
 3     if len(tour)==len(graphe):
 4         return tour  # tour complet
 5
 6     min_dist = math.inf; candidat = None
 7     for n in graphe[v]:
 8         if not n in tour:
 9             if graphe[v][n]<min_dist:
10                 min_dist = graphe[v][n]
11                 candidat = n
12
13     ClstNeighbor_TSP(graphe, candidat)
14
15 tour = [] # pour recolter l'ordre des sommets
16 ClstNeighbor_TSP(graphe,depart_arbitraire)
```

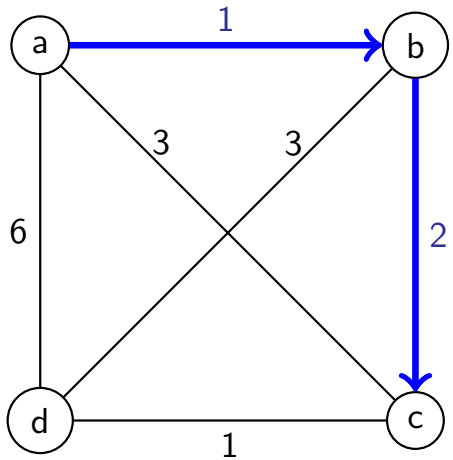
Solution de “le plus proche voisin”



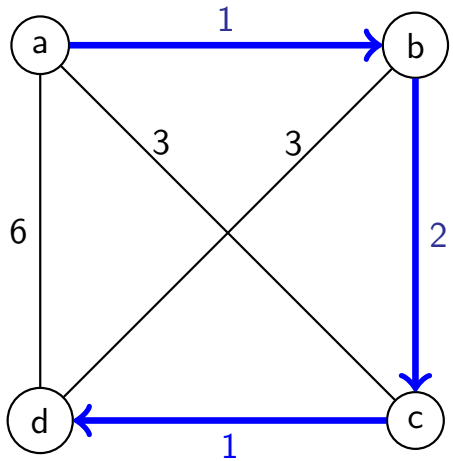
Solution de “le plus proche voisin”



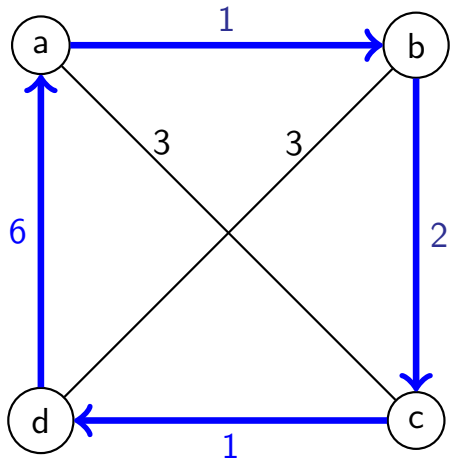
Solution de “le plus proche voisin”



Solution de “le plus proche voisin”

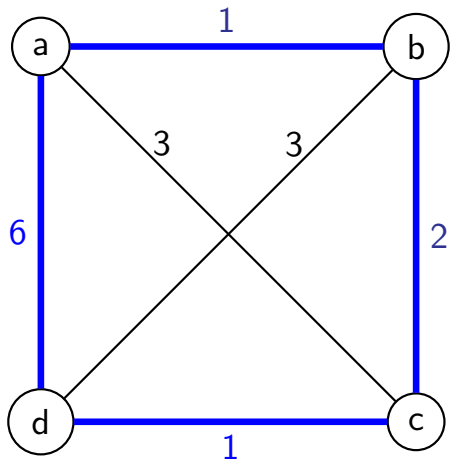


Solution de “le plus proche voisin”



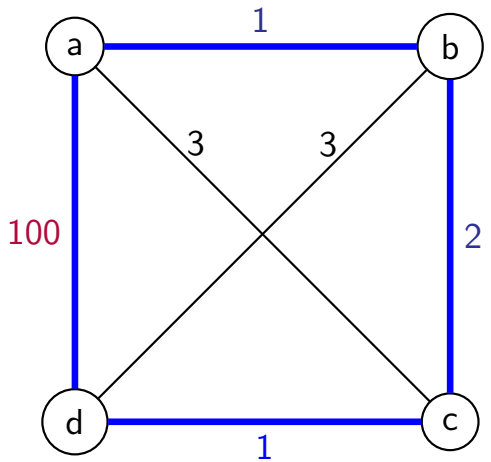
Solution *greedy* = 1+2+1+**6**=10

Qualité de la solution de “le plus proche voisin”



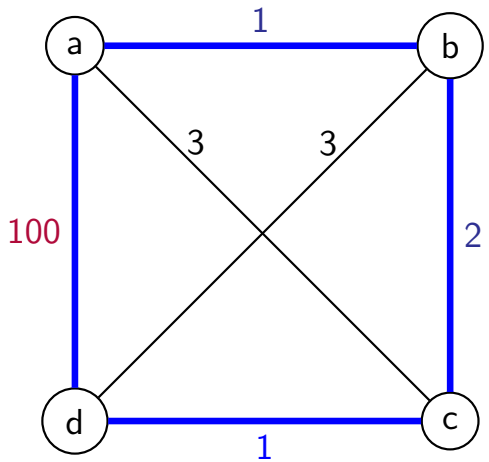
Solution *greedy* = 1+2+1+**6**=10

Qualité de la solution de “le plus proche voisin”



Solution *greedy* = 1+2+1+**100**=104

Qualité de la solution de “le plus proche voisin”

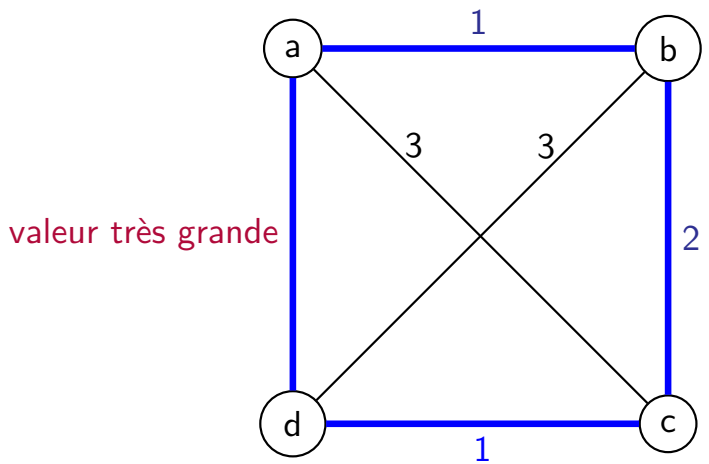


Solution *greedy* = 1+2+1+**100**=104

La dernière arête choisie détermine la qualité de la solution

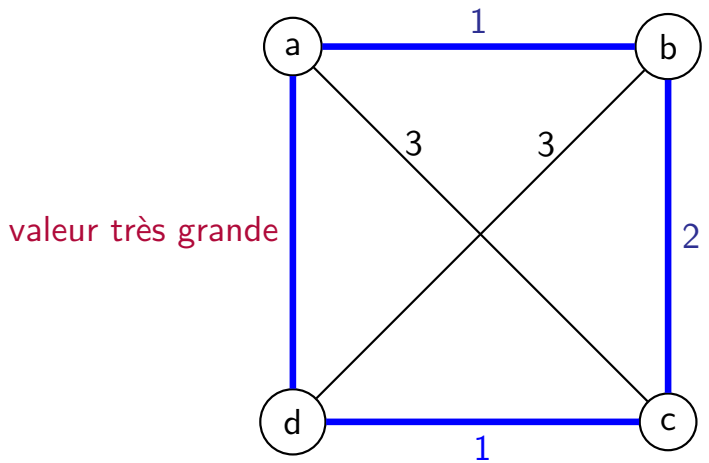
En remplaçant la distance **6** par une valeur très grande, nous dégradons la qualité de la solution.

Qualité de la solution de “le plus proche voisin”



Solution *greedy* = 1+2+1+**valeur très grande** = **valeur très grande**

Qualité de la solution de “le plus proche voisin”



Solution *greedy* = 1+2+1+**valeur très grande** = **valeur très grande**

Conclusion

La qualité d’une solution produite par notre algoritme n’est pas garantie.

Qualité de solution

Heuristique

Un algorithme heuristique produit une solution d'un problème difficile en temps polynomial, mais il ne garantit pas sa qualité.

Qualité de solution

Heuristique

Un algorithme heuristique produit une solution d'un problème difficile en temps polynomial, mais il ne garantit pas sa qualité.

Algorithme d'approximation

Un algorithme d'approximation produit une solution d'un problème difficile **en temps polynomial** dont la qualité est connue. On sait **combien de fois au pire** la solution produite pour un problème de

- **minimisation** est **plus grande** que la solution optimale
- **maximisation** est **plus petite** que la solution optimale.

Pour donner une idée...

...de la manière de procéder nous vous donnerons un algorithme d'approximation pour résoudre TSP.

Pour donner une idée...

...de la manière de procéder nous vous donnerons un algorithme d'approximation pour résoudre TSP.

Nous nous limitons aux **espaces métriques**

où **l'inégalité triangulaire est valide** et nous procédons par construction *.

*. Non, ce ne sera pas un glouton cette fois !

Pour donner une idée...

...de la manière de procéder nous vous donnerons un algorithme d'approximation pour résoudre TSP.

Nous nous limitons aux **espaces métriques**

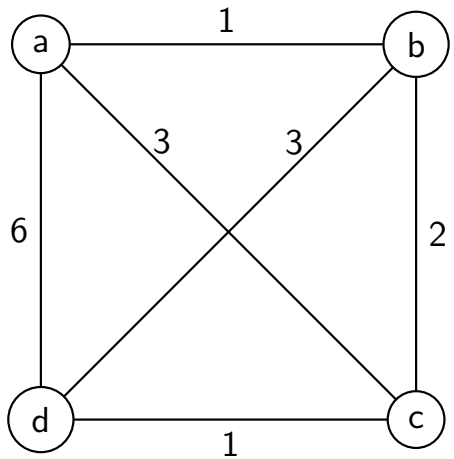
où **l'inégalité triangulaire est valide** et nous procédons par construction *.

*. Non, ce ne sera pas un glouton cette fois !

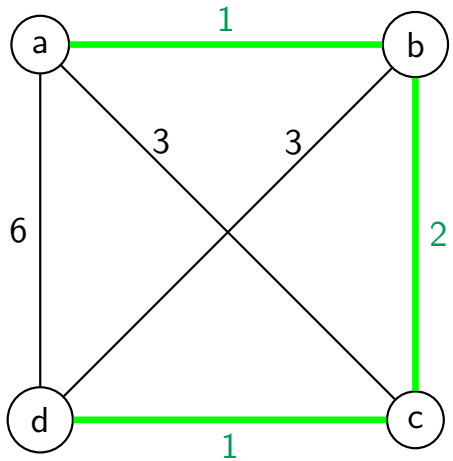
Avant de commencer

Notre algorithme est en trois étapes. Dans leur description OPT signifiera la longueur d'une **solution optimale** de TSP que **nous ne connaissons pas**.

Première étape : Etablir une solution T de MST pour G



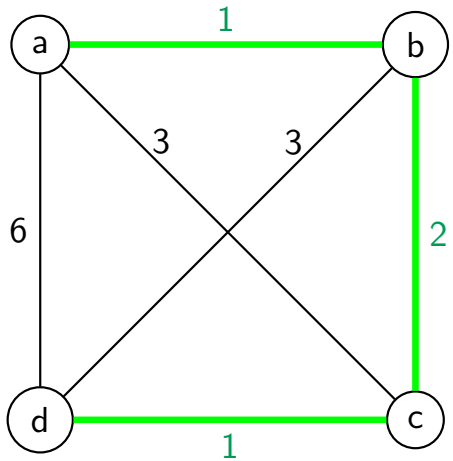
Première étape : Etablir une solution T de MST pour G



Observation

$$\text{poids}(T) \leq \text{OPT}$$

Première étape : Etablir une solution T de MST pour G



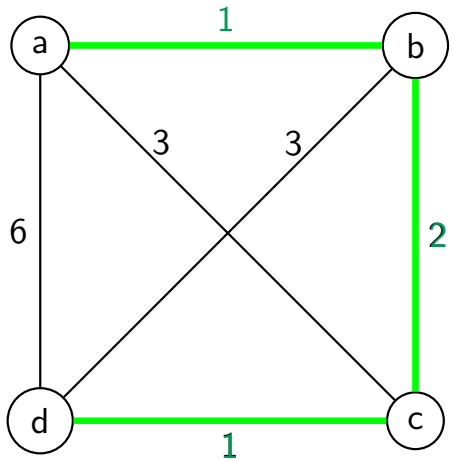
Observation

$$\text{poids}(T) \leq \text{OPT}$$

voir cours 3 sur MST : le poids d'un MST ne sera jamais plus grand que celui d'une solution optimale de TSP étant un cycle

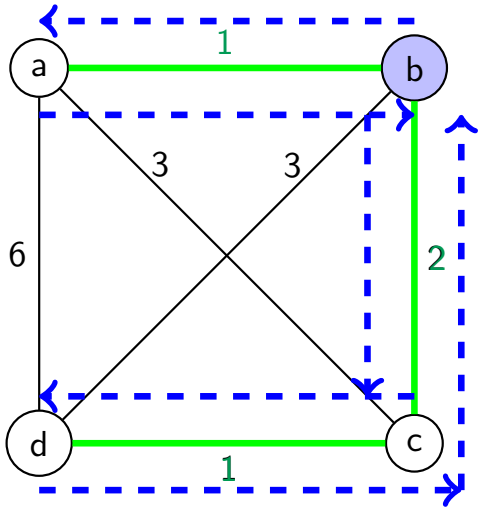
Complexité : polynomiale (la même que Kruskal/Prim)

Deuxième étape : Exécuter un DFS de T



T_D est un parcours en profondeur de T tel que chaque arête de T est traversée **deux** fois. En partant de b : $T_D = (b, a, b, c, d, c, b)$.

Deuxième étape : Exécuter un DFS de T

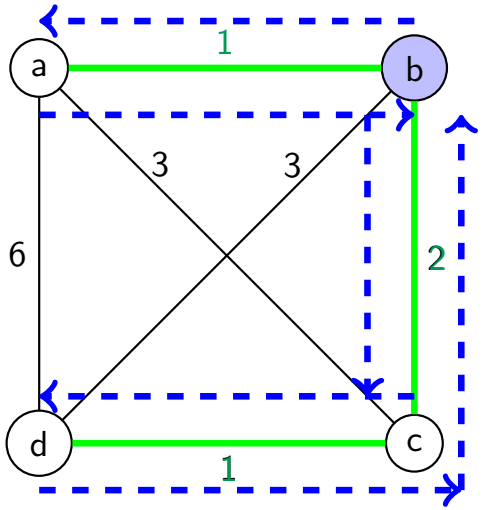


T_D est un parcours en profondeur de T tel que chaque arête de T est traversée **deux** fois. En partant de b : $T_D = (b, a, b, c, d, c, b)$.

Observation

$$\text{longueur}(T_D) \leq 2\text{OPT}$$

Deuxième étape : Exécuter un DFS de T



T_D est un parcours en profondeur de T tel que chaque arête de T est traversée **deux** fois. En partant de b : $T_D = (b, a, b, c, d, c, b)$.

Observation

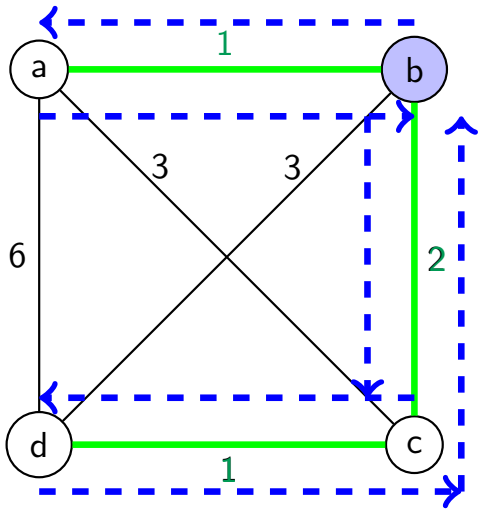
$$\text{longueur}(T_D) \leq 2\text{OPT}$$

La longueur du parcours de T_D est au plus **deux** fois plus grande que le poids de T.

Complexité : polynomiale (la même que du DFS).

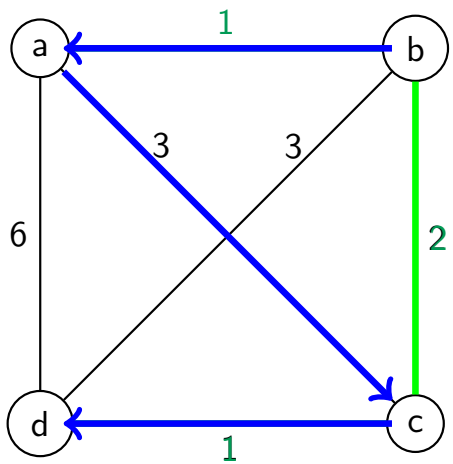
Troisième étape : Transformer T_D en cycle C

Garder dans T_D la première occurrence d'un sommet seulement :
$T_D = (b, a, b, c, d, c, b) \rightarrow (b, a, \cancel{b}, c, d, \cancel{c}, \cancel{b}) \rightarrow (b, a, c, d) = C$.



Troisième étape : Transformer T_D en cycle C

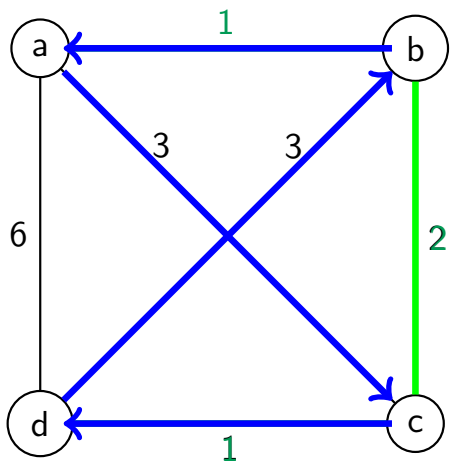
Garder dans T_D la première occurrence d'un sommet seulement :
$T_D = (b, a, b, c, d, c, b) \rightarrow (b, a, \cancel{b}, c, d, \cancel{c}, \cancel{b}) \rightarrow (b, a, c, d) = C$.



Observation

Troisième étape : Transformer T_D en cycle C

Garder dans T_D la première occurrence d'un sommet seulement :
$T_D = (b, a, b, c, d, c, b) \to (b, a, \cancel{b}, c, d, \cancel{c}, \cancel{b}) \to (b, a, c, d) = C$.

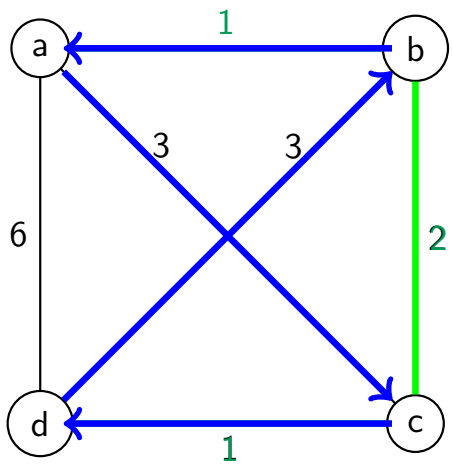


Observation

Grâce à l'inégalité triangulaire : longueur$(C) \leq$ longueur(T_D)

Troisième étape : Transformer T_D en cycle C

Garder dans T_D la première occurrence d'un sommet seulement :
$T_D = (b, a, b, c, d, c, b) \to (b, a, \cancel{b}, c, d, \cancel{c}, \cancel{b}) \to (b, a, c, d) = C$.

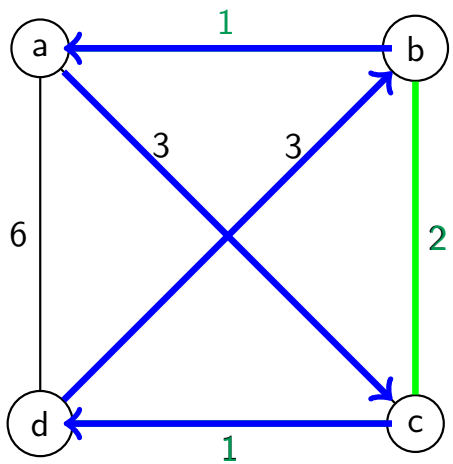


Observation

Grâce à l'inégalité triangulaire : longueur$(C) \leq$ longueur(T_D)
par conséquent : longueur$(C) \leq$ longueur$(T_D) \leq$ **2**OPT.

Troisième étape : Transformer T_D en cycle C

Garder dans T_D la première occurrence d'un sommet seulement :
$T_D = (b, a, b, c, d, c, b) \to (b, a, \cancel{b}, c, d, \cancel{c}, \cancel{b}) \to (b, a, c, d) = C$.



Observation

Grâce à l'inégalité triangulaire : longueur$(C) \leq$ longueur(T_D)
par conséquent : longueur$(C) \leq$ longueur$(T_D) \leq$ **2**OPT.
Complexité : polynomiale (linéaire en $|V|$).

Pour en finir avec TSP

Récapitulatif :

- notre algorithme d'approximation produit une solution jamais **deux** fois plus longue qu'une solution optimale dans un espace où l'inégalité triangulaire s'applique,
- il existe un algorithme d'approximation utilisant l'inégalité triangulaire qui rend une solution jamais $\frac{3}{2}$ fois plus longue qu'une solution optimale (l'algorithme de Christofides : construction en cinq étapes dont la première est aussi la solution du MST),
- dans des espaces non-métriques il est impossible de trouver un algorithme d'approximation pour TSP.

Approximation : formellement

Soient $\mathcal{P}$ un problème d'optimisation et f la fonction d'évaluation des solutions de $\mathcal{P}$ et $\mathcal{A}$ un algorithme d'approximation.

Approximation : formellement

Soient $\mathcal{P}$ un problème d'optimisation et f la fonction d'évaluation des solutions de $\mathcal{P}$ et $\mathcal{A}$ un algorithme d'approximation.

Définition

Soient I une instance de $\mathcal{P}$, S une solution pour I produite par $\mathcal{A}$ et S^* une solution optimale pour I.

- le rapport d'approximation de S sur I est : $\rho(I, S) = \frac{f(S)}{f(S^*)}$
- l'algorithme $\mathcal{A}$ est une ρ-approximation pour $\mathcal{P}$ si et seulement si : $\rho(I, S) \leq \rho$

Pour et contre un algorithme d'approximation

Pour et contre un algorithme d'approximation

- ✓ Temps polynomial
- ✓ Garantie sur la qualité de solution
- ✓ En pratique, une solution obtenue potentiellement meilleure que la garantie pourrait le faire croire
- ✗ Impossibilité de trouver un tel algorithme *
- ✗ Temps polynomial, mais exécution en pratique trop longue §
- ✗ Difficulté d'implémentation §
- ✗ Solutions produites "assez loin" d'une solution exacte §

∗. pour certains problèmes ils n'existent pas

§. possible

Plan

1 Problème du Voyageur de Commerce

2 Méthodes exactes

3 Heuristiques et approximation

4 Conclusion

Ce qu'il faut retenir

Problème d'optimisation NP-difficile...

- Solution exacte uniquement pour des instances de petite taille
  car la complexité d'un algorithme exact est toujours exponentielle
- Instances de grandes tailles $\rightarrow$ une heuristique ou une méthode d'approximation
  ➜ On cherchera en temps polynomial des **solutions "qui ne sont pas si mauvaises que ça"** (mais elles ne sont pas nécessairement optimales).