Méthodes arborescentes exactes et approchées

Complexité, Algorithmes Randomisés et Approchés

Problème d'optimisation

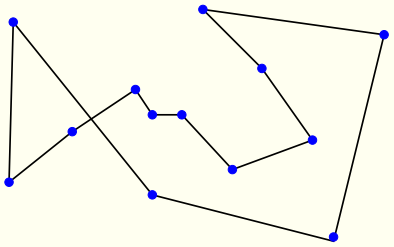
Définition :

- Nom du problème : P
- Paramètres génériques du problème (nombres, graphes, ...)
- Une caractérisation de ce qu'est une solution réalisable :
  - Une instance I de P est une instanciation des paramètres génériques
  - A chaque I correspond un ensemble de solutions réalisables $S(I)$.
- Une fonction objectif f

Résolution :
Déterminer un algorithme A qui, pour chaque instance I retourne une solution $s^*(I)$ de $S(I)$ t.q. :

problème de minimisation : $\forall s \in S(I), f(s^*(I)) \leq f(s)$, ou
problème de maximisation : $\forall s \in S(I), f(s^*(I)) \geq f(s)$.

Exemple : le voyageur de commerce euclidien (TSP)



Données : n points dans le plan
Solution réalisable : Un cycle hamiltonien dans le graphe complet sous-jacent K_n
Fonction objectif : La longueur du cycle (que l'on souhaite minimiser)

Le problème de décision associé à ce problème est NP-complet.
$\Rightarrow$ ce problème est NP-difficile.

Trouver une solution optimale du TSP

Première idée : énumérer l'ensemble des solutions réalisables du TSP : $\frac{(n-1)!}{2}$ cycles possibles

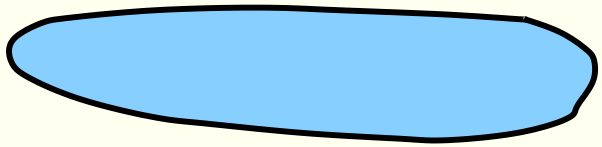
But : Trouver une solution optimale sans énumérer toutes les solutions.

“Branch-and-bound” (séparation-évaluation)

- “Branch” (brancher)
  - Diviser (partitionner) l’espace de recherche
    → Arbre d’énumération (ou arbre de recherche)
  - Explorer l’arbre de recherche

Brancher

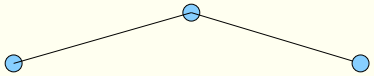
La racine de l’arbre représente l’ensemble des solutions.
Chaque sous-arbre représente une solution partielle.

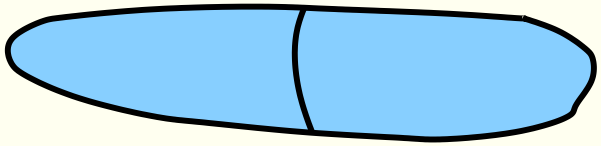


Espace des solutions

Brancher

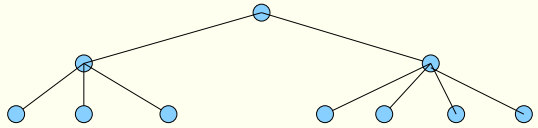
La racine de l’arbre représente l’ensemble des solutions.
Chaque sous-arbre représente une solution partielle.

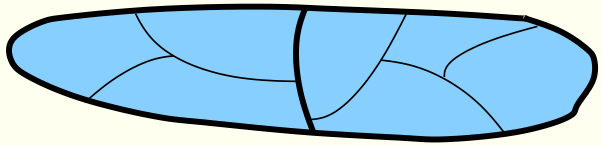




Brancher

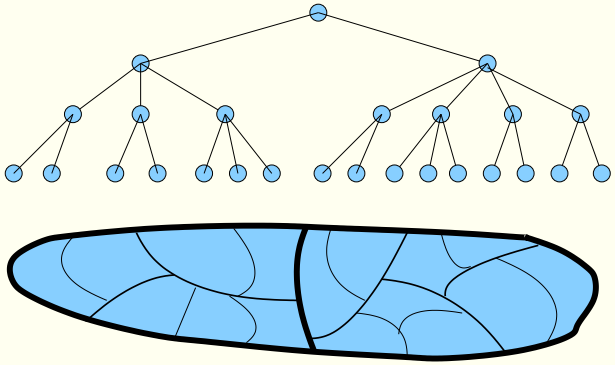
La racine de l’arbre représente l’ensemble des solutions.
Chaque sous-arbre représente une solution partielle.





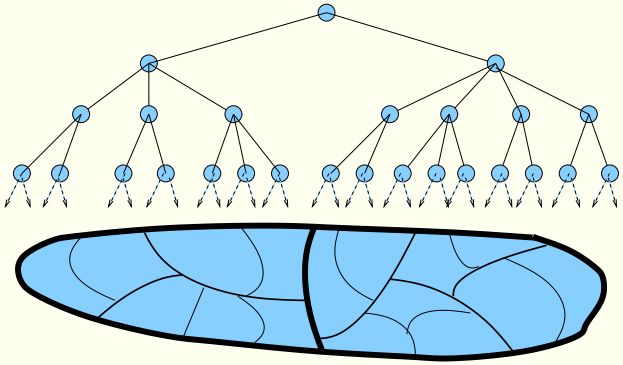
Brancher

La racine de l'arbre représente l'ensemble des solutions.
Chaque sous-arbre représente une solution partielle.



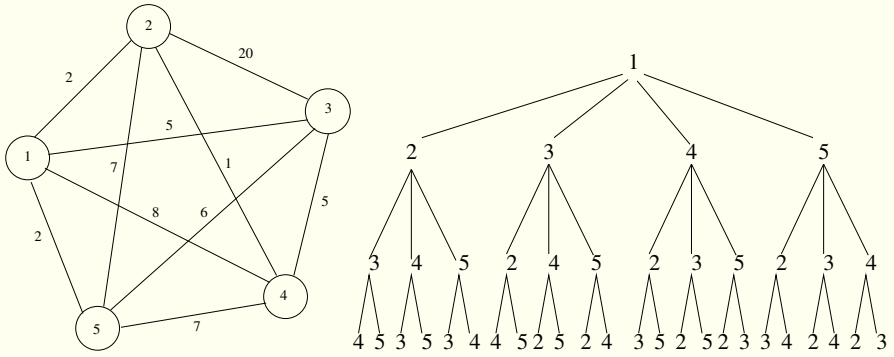
Brancher

La racine de l'arbre représente l'ensemble des solutions.
Chaque sous-arbre représente une solution partielle.



Brancher : exemple sur le problème du TSP

Soit 1 le premier sommet du cycle.
On choisit au niveau i le i^{eme} sommet du cycle ($n - i$ choix).



"Branch-and-bound" (séparation-évaluation)

- "Branch" (brancher)
  - Diviser (partitionner) l'espace de recherche
    $\rightarrow$ Arbre d'énumération (ou arbre de recherche)
  - Explorer l'arbre de recherche
- "Bound" (borner) (présenté pour un pb de minimisation)
  - Borne supérieure de la valeur d'une solution optimale
  - Borne inférieure de la valeur d'un nœud (et de son sous-arbre)

Borner

Au noeud courant on a :

- une borne supérieure B_{sup} d'une solution optimale. C'est souvent le coût d'une solution réalisable que l'on a déjà rencontrée.
  $\rightarrow$ ce que l'on a déjà
- une borne inférieure B_{inf} du coût de toute solution issue du noeud courant (borne inf de toute solution du sous-arbre courant)
  $\rightarrow$ ce que l'on peut espérer avoir de mieux en explorant le sous-arbre

Si $B_{inf} > B_{sup}$ alors on "élague" : on n'explore pas le sous-arbre enraciné au noeud courant.

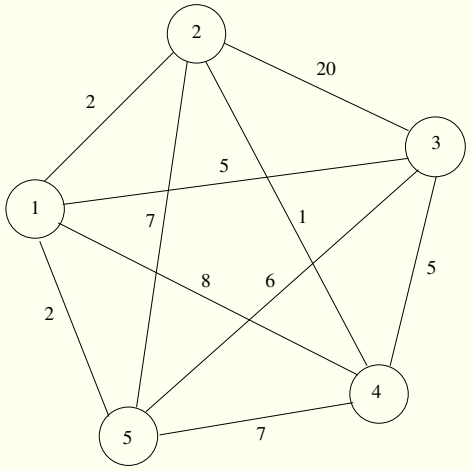
Borner : exemple sur le problème du TSP

Soit $G = (S, A)$ un graphe valué. Soit $i \in S$ un sommet. Soit $min_1(i)$ le coût de la plus petite arête adjacente à i et $min_2(i)$ le coût de la 2ème plus petite arête adjacente à i.

Propriété : Le coût d'un cycle hamiltonien de G est $\geq \frac{1}{2} \sum_{i=1}^{n} min_1(i) + min_2(i)$.

Borne inférieure du coût d'une solution dont les sommets S' forment un cycle partiel $(s_1, \ldots, s_k)$ = coût des arêtes du cycle partiel $+\frac{1}{2}(min_1(s_1) + min_1(s_k)) + \frac{1}{2} \sum_{i \in S \setminus S'} (min_1(i) + min_2(i))$

Exemple



"Branch-and-bound" (séparation-évaluation)

Un algorithme de branch-and-bound pour un problème de minimisation est basé sur

- une procédure de branchement qui décompose le problème,

  et

- une borne inférieure pour éviter d'avoir à parcourir tout l'arbre.

Arbre d'énumération

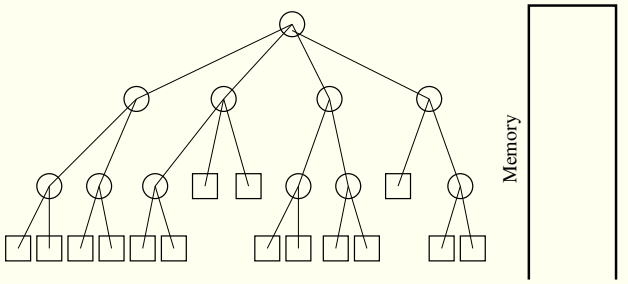
- L'arbre d'énumération n'est pas complètement stocké en mémoire.
- En effet, sa taille est proportionnelle à la taille de l'ensemble des solutions, qui est exponentielle.
- L'arbre d'énumération est exploré pour trouver la solution optimale.

Il y a deux façons classiques d'explorer l'arbre :

- Parcours en profondeur
- Parcours "le meilleur d'abord"

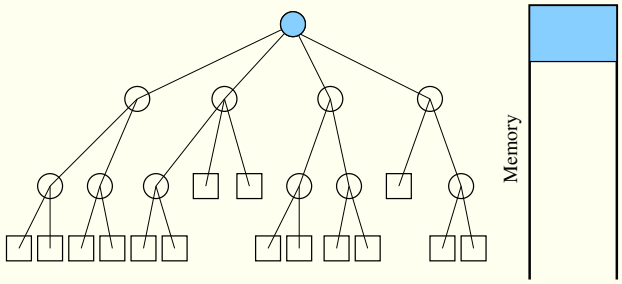
Explorer l'arbre d'énumération

Parcours en profondeur



Explorer l'arbre d'énumération

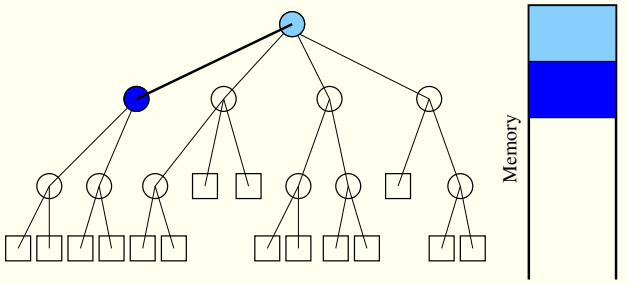
Parcours en profondeur



Charger le problème en mémoire

Explorer l'arbre d'énumération

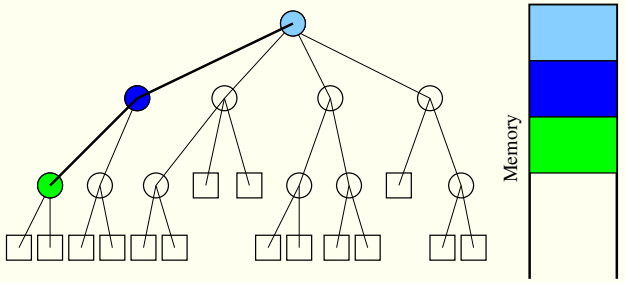
Parcours en profondeur



Première branche = premier sous-problème

Explorer l'arbre d'énumération

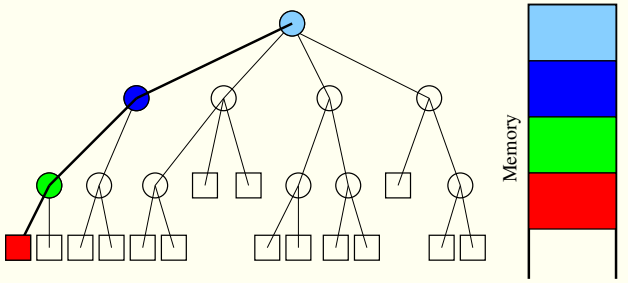
Parcours en profondeur



Descendre

Explorer l'arbre d'énumération

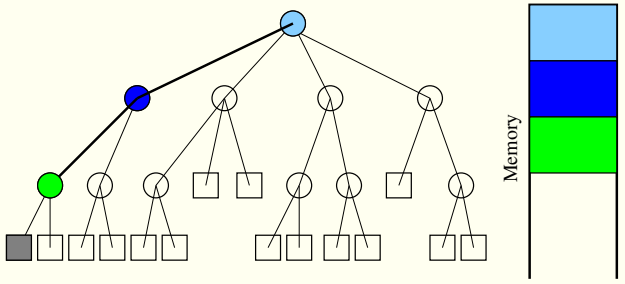
Parcours en profondeur



Première feuille

Explorer l'arbre d'énumération

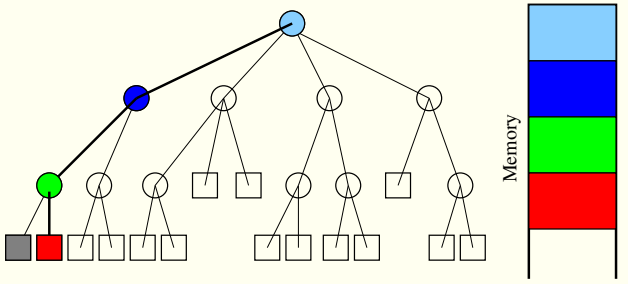
Parcours en profondeur



Premier retour en arrière

Explorer l'arbre d'énumération

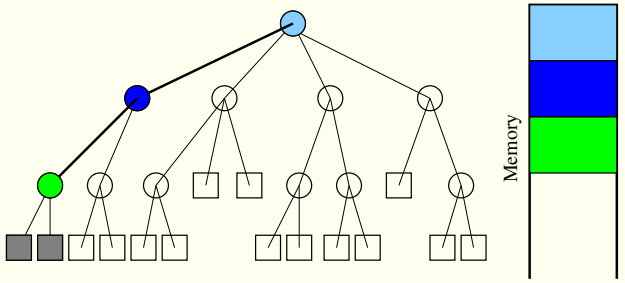
Parcours en profondeur



Seconde feuille

Explorer l'arbre d'énumération

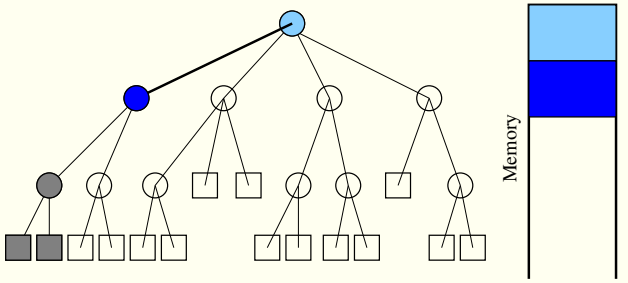
Parcours en profondeur



Deuxième retour en arrière

Explorer l'arbre d'énumération

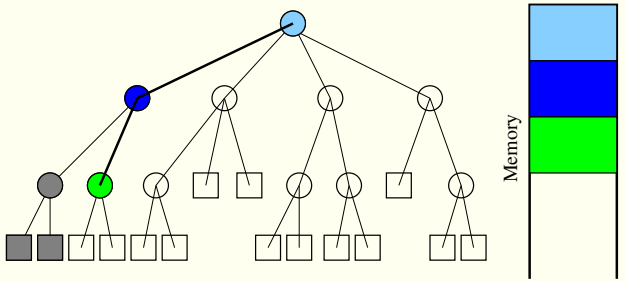
Parcours en profondeur



et retourner en arrière à nouveau

Explorer l'arbre d'énumération

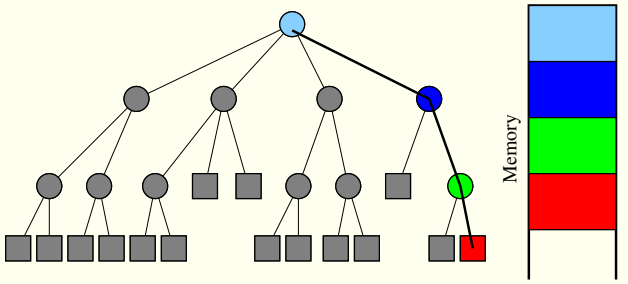
Parcours en profondeur



et ainsi de suite...

Explorer l'arbre d'énumération

Parcours en profondeur



...jusqu'à la dernière feuille

Complexité

- Complexité temporelle : généralement exponentielle en la taille du problème.
- Complexité en espace : en $O(hn)$ avec n taille du problème et h hauteur de l'arbre d'énumération (h est polynomial en n).
- Les feuilles intéressantes traversées pendant la recherche doivent être stockées.
  - En général on stocke seulement une solution : la meilleure solution rencontrée (ou l'une parmi les meilleures).
    La taille de la solution est $O(n)$.

Explorer l'arbre d'énumération

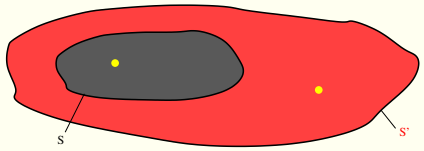
Parcours "le meilleur d'abord"

- A chaque itération, parmi les sommets ouverts, on choisit "le plus prometteur" (par exemple celui qui a la borne inférieure la plus basse).
- Les sommets ouverts sont stockés dans un tas (priorité d'un noeud = sa borne inférieure).
- Inconvénient : la taille du tas peut être grande.

Comment trouver une borne inférieure ?

- Coût de la solution partielle
- Solution "ad hoc" (propriété)
- Relaxation du problème

Trouver une borne inférieure : relaxation du problème



- Ignorer certaines contraintes définissant S, ou rendre ces contraintes moins fortes.
- Soit S' le nouvel ensemble de solutions ($S \subset S'$).
- Le minimum du problème relâché est une borne inférieure. $\min_{x \in S'} f(x) \leq \min_{x \in S} f(x)$.
- On cherche un problème relâché qui est polynomial.

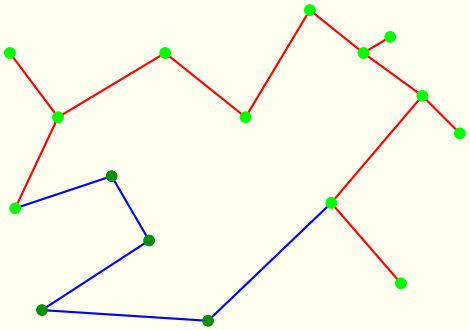
Exemple : TSP

- Une chaîne hamiltonienne (CH) est une châine qui passe exactement une fois par chaque sommet du graphe.
- Une CH est un arbre couvrant avec la contrainte additionnelle que l'arbre doit avoir seulement deux feuilles.
- Le problème de l'arbre couvrant de poids minimum (ACPM) est une relaxation du problème de la CH la plus courte. Le coût d'un ACPM est une borne inférieure de la CH la plus courte, qui est une borne inférieure pour le TSP.

TSP : borne inf pour une solution partielle

Coût d'une solution partielle $\geq$

- coût du tour partiel
- $+$ coût d'un arbre couvrant de poids minimum pour les sommets qui ne sont pas couverts par le tour partiel



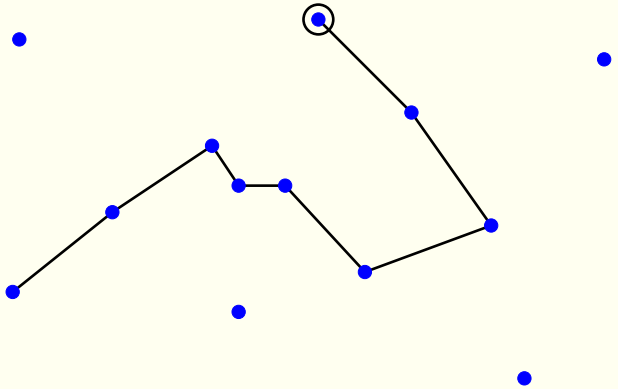
En pratique

- Plus la borne inférieure est bonne, plus le nombre de nœuds visités pendant la recherche est faible.
- Cependant, si la borne inférieure est meilleure, son temps de calcul peut être plus long
  - moins de nœuds visités
  - plus de temps passé à chaque nœud
  - il faut faire des tests pour savoir quelle solution est la meilleure.
- Une bonne solution initiale est très importante.

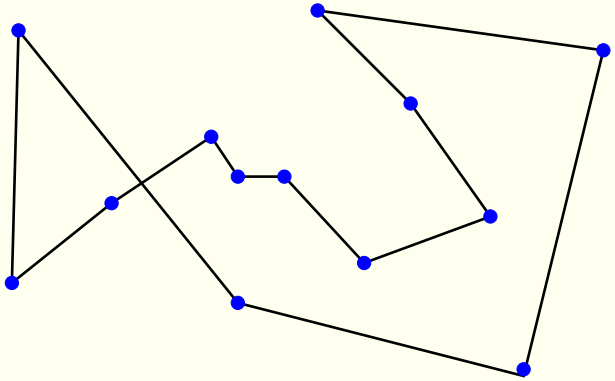
Construire une bonne solution

- 1. Algorithmes gloutons
  Construire une solution à partir de règles simples
- 2. Heuristiques basées sur une relaxation du problème
  Construire une solution réalisable à partir d'une solution relâchée qui viole certaines contraintes
- 3. Branch and bound partiel
  Utiliser un arbre d'énumération pour obtenir une bonne solution sans explorer tout l'espace des solutions.

Règle du PlusProcheVoisin pour le TSP

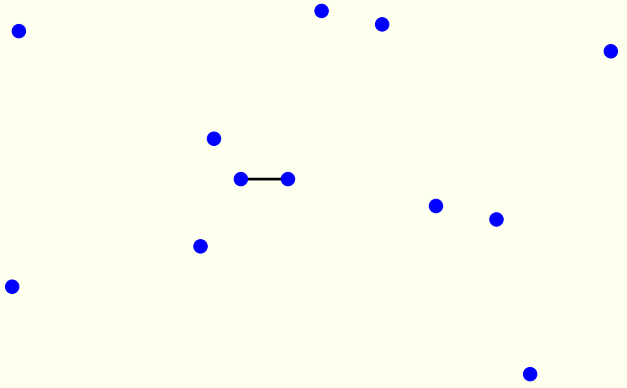


Règle du PlusProcheVoisin pour le TSP



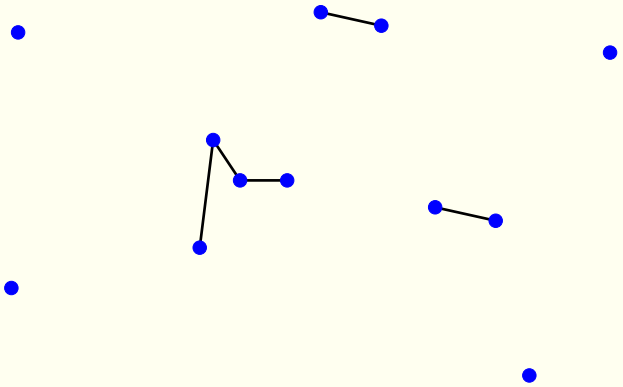
La solution est réalisable mais pas optimale

Une autre heuristique gloutonne



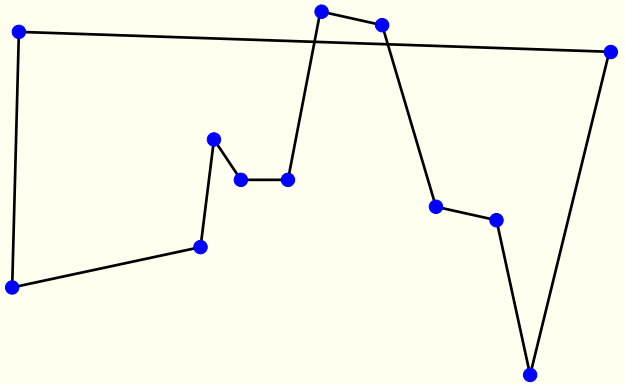
Comme dans l'algorithme de Kruskal, connecter les points les plus proches.

Une autre heuristique gloutonne



L'arête la plus courte qui ne crée pas de nœud de degré 3 ni de cycle est choisie.

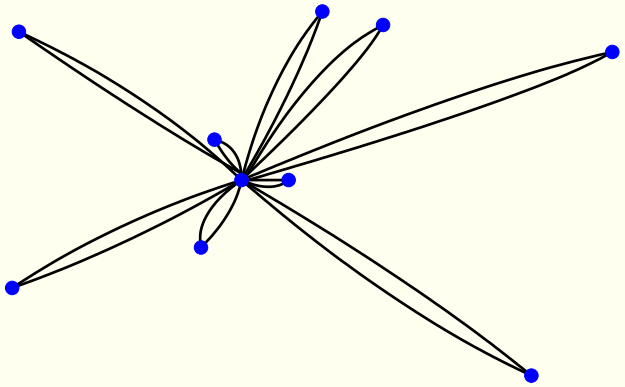
Une autre heuristique gloutonne



Heuristique de Clarke-Wright

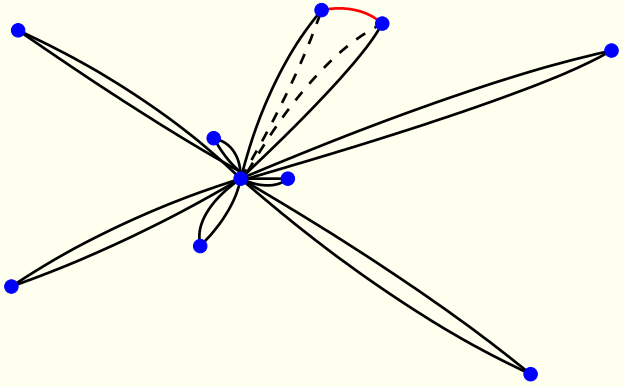
- Cette heuristique a été initialement introduite pour le problème de tournées de véhicules (VRP).
- Le VRP est une généralisation du TSP dans laquelle
  - il y a un dépôt à partir duquel tous les tours commencent et se terminent.
  - les véhicules ont une capacité limitée : les livraisons doivent être divisées en plusieurs tournées.
- Pour construire un tour pour le TSP, on choisit arbitrairement une ville comme étant le dépôt et les véhicules ont une capacité illimitée.

Heuristique de Clarke-Wright



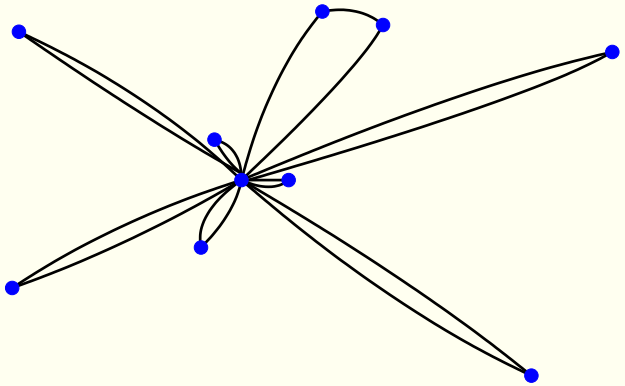
Créer un tour individuel entre chaque point et le dépôt.

Heuristique de Clarke-Wright



Le gain d'une arête mesure la diminution de la longueur du tour obtenu en choisissant cette arête.

Heuristique de Clarke-Wright

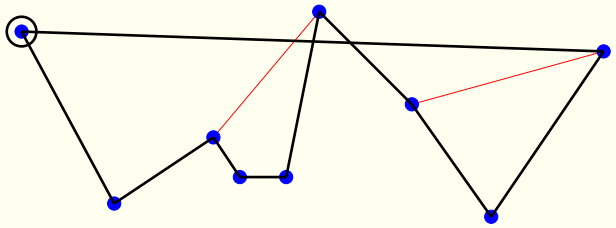


L'arête avec le gain le plus important est choisie à chaque étape.

2. Heuristiques basées sur la relaxation du problème

- Trouver une relaxation du problème
  Le TSP euclidien est relaxé en un arbre couvrant de poids minimum
- Retourner une solution optimale pour le problème relâché
  Algorithme de Prim ou de Kruskal
- Si la solution du problème relâché viole certaines contraintes du problème initial, réparer cette solution pour la rendre réalisable.

Construire un tour à partir d'un arbre couvrant de poids minimum

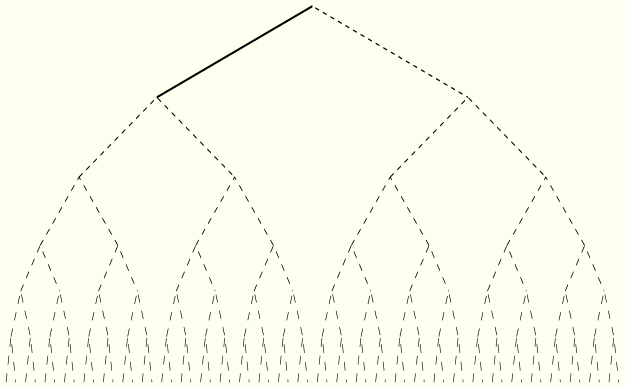


3. Recherche partielle dans un arbre d'énumération

- Le but est d'utiliser une structure générale (l'arbre d'énumération) afin d'avoir des algorithmes génériques.
- Au lieu de construire une seule solution, on construit plusieurs solutions en visitant plusieurs feuilles de l'arbre.
- On a besoin d'une heuristique pour évaluer *à priori* les différents choix représentés par les différentes branches descendant d'un nœud.

Heuristique gloutonne dans un arbre de recherche

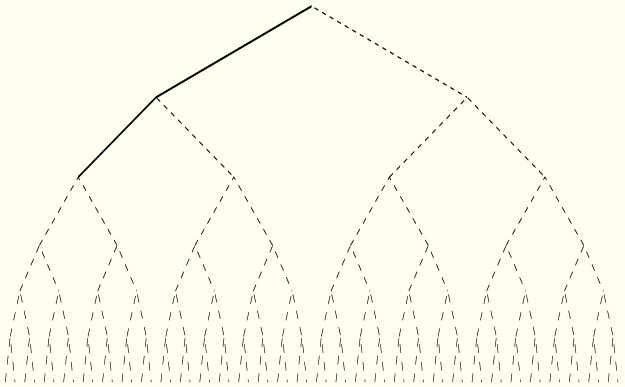
La meilleure branche est dessinée comme étant la branche gauche.



Choisir la meilleure branche à priori

Heuristique gloutonne dans un arbre de recherche

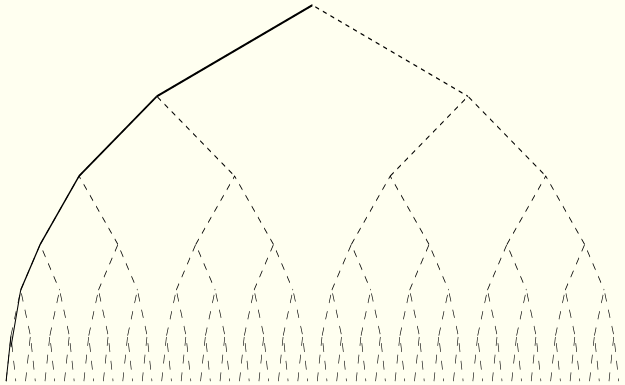
La meilleure branche est dessinée comme étant la branche gauche.



et répéter le même processus de sélection

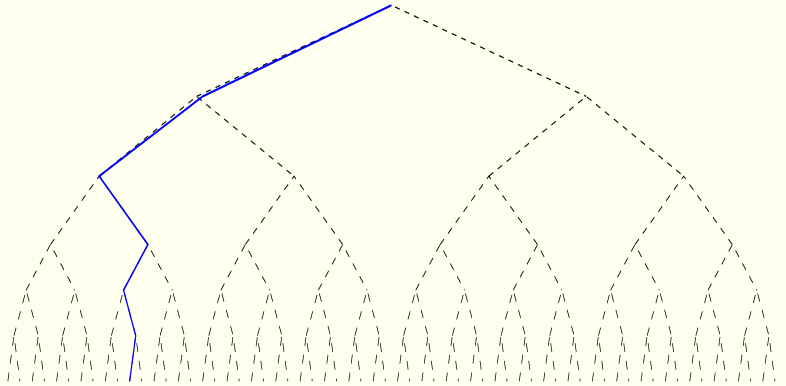
Heuristique gloutonne dans un arbre de recherche

La meilleure branche est dessinée comme étant la branche gauche.



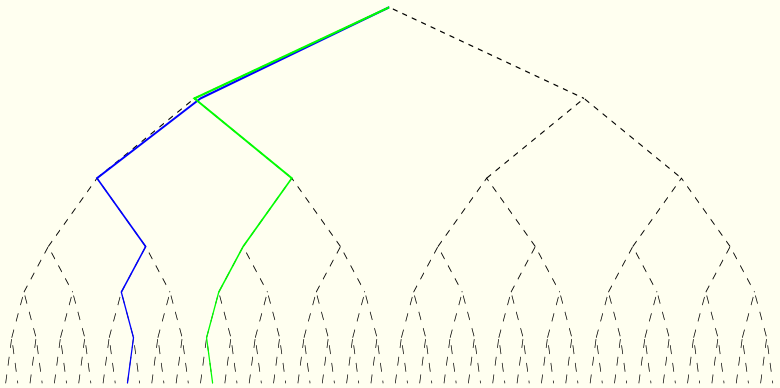
jusqu'à trouver la solution

Greedy Randomized Adaptive Search Procedure



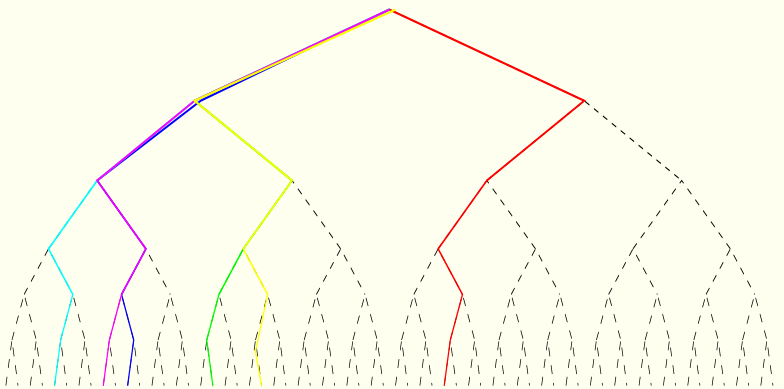
Choisir la branche gauche avec la probabilité $1 - p$ avec $p < 0.5$

Greedy Randomized Adaptive Search Procedure



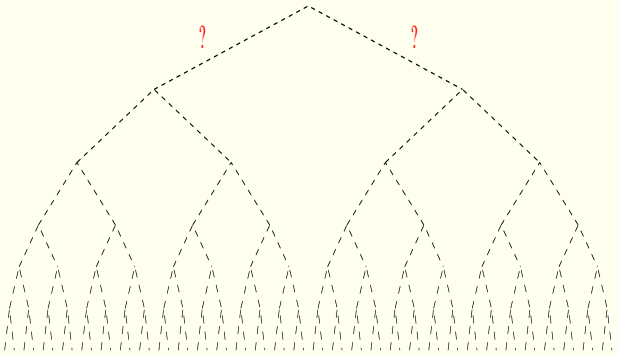
Recommencer la procédure

Greedy Randomized Adaptive Search Procedure



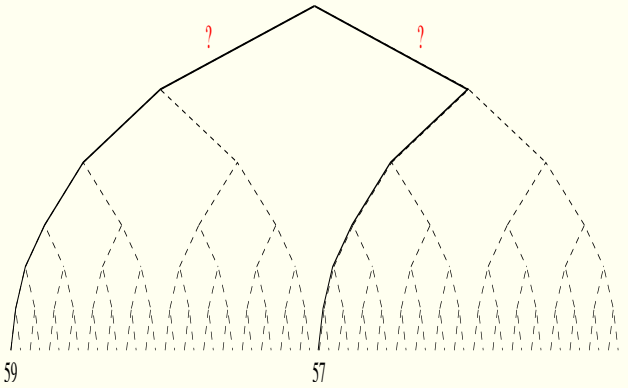
Et choisir la meilleure

Branch-and-Greed



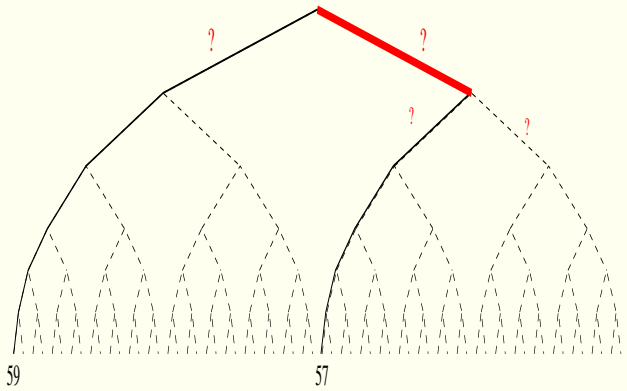
Comment choisir entre les deux premières branches du nœud racine?

Branch-and-Greed



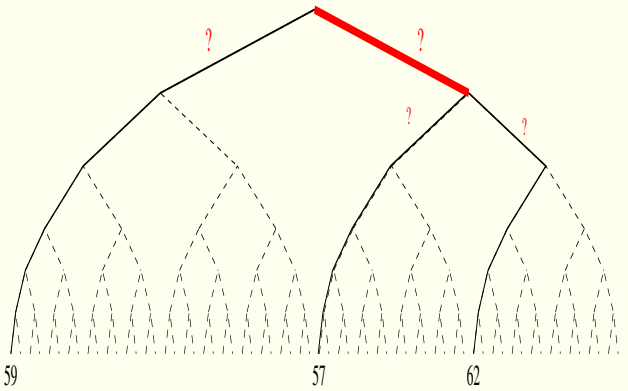
Exécuter l'algorithme glouton pour chaque sous-arbre

Branch-and-Greed



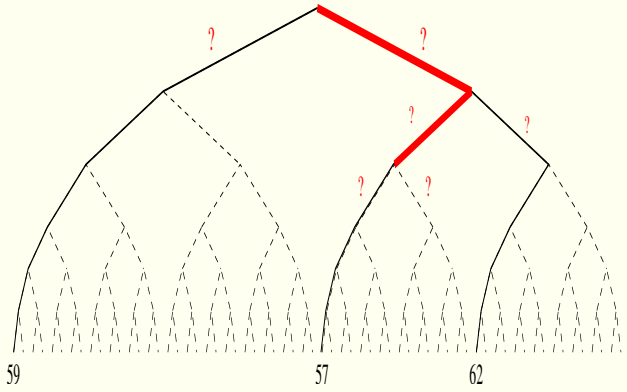
et sélectionner la branche qui mène au sous-arbre avec la meilleure solution.

Branch-and-Greed



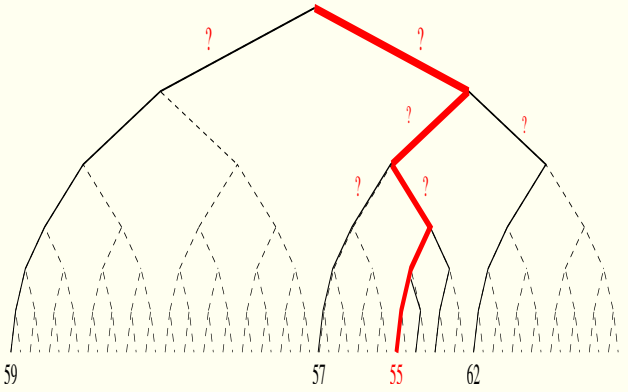
Exécuter à nouveau l'algorithme glouton pour chaque sous-arbre du sous-arbre courant

Branch-and-Greed



Emprunter la meilleure branche

Branch-and-Greed



Jusqu'à ce que le chemin rouge atteigne une feuille

Résumé : "Branch-and-bound" (séparation-évaluation)

- "Branch" (brancher)
  - Diviser (partitionner) l'espace de recherche
    $\rightarrow$ Arbre d'énumération (ou arbre de recherche)
  - Explorer l'arbre de recherche
- "Bound" (borner)
  - Borne supérieure de la valeur d'une solution optimale (solution réalisable)
  - Borne inférieure de la valeur d'un nœud

Pour réduire (encore) le nombre de noeuds explorés : ensemble dominant, règle de dominance.

Ensemble dominant

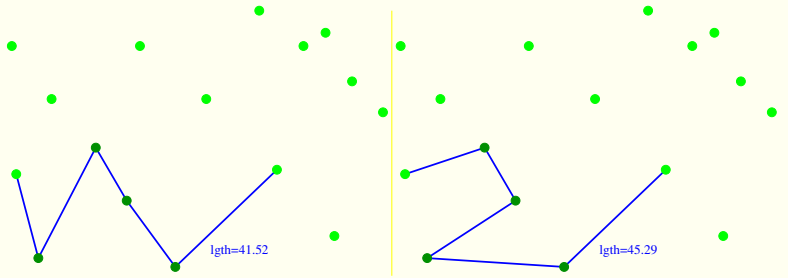
- Pour certains problèmes, on peut facilement déterminer un sous-ensemble S' de solutions de S tel que S' contient au moins une solution optimale de S.
- L'algorithme de branch-and-bound doit explorer S' au lieu de S.

Règle de dominance

- On peut souvent prouver une propriété (P) qui doit être vérifiée par au moins une solution optimale du problème.
- En analysant une solution partielle à un nœud de l'arbre de recherche, on peut quelquefois prouver que (P) ne peut pas être vérifiée par les solutions qui complètent la solution partielle courante.
- Le nœud est dominé par une solution partielle qui satisfait (P).
- La propriété (P) est appelée règle de dominance.

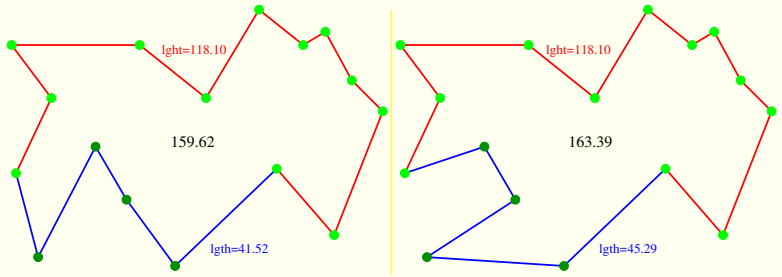
Une règle de dominance pour le TSP

Le tour partiel (chemin) $(s_1 = 1, s_2, \cdots, s_k)$ doit être un chemin hamiltonien entre s_1 et s_k dans le sous-graphe défini par les nœuds $\{s_1, s_2, \ldots, s_k\}$.



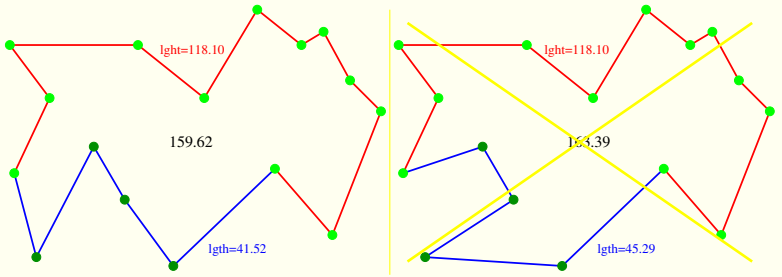
Une règle de dominance pour le TSP

Le tour partiel (chemin) $(s_1 = 1, s_2, \cdots, s_k)$ doit être un chemin hamiltonien entre s_1 et s_k dans le sous-graphe défini par les nœuds $\{s_1, s_2, \ldots, s_k\}$.



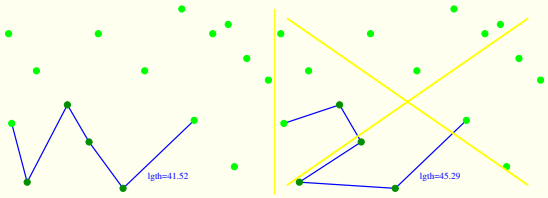
Une règle de dominance pour le TSP

Le tour partiel (chemin) $(s_1 = 1, s_2, \cdots, s_k)$ doit être un chemin hamiltonien entre s_1 et s_k dans le sous-graphe défini par les nœuds $\{s_1, s_2, \ldots, s_k\}$.



Une règle de dominance pour le TSP

Le tour partiel (chemin) $(s_1 = 1, s_2, \cdots, s_k)$ doit être un chemin hamiltonien entre s_1 et s_k dans le sous-graphe défini par les nœuds $\{s_1, s_2, \ldots, s_k\}$.



Un tour partiel est dominé quand on peut montrer qu'il existe un tour partiel plus court.