Voyageur de commerce (TSP)

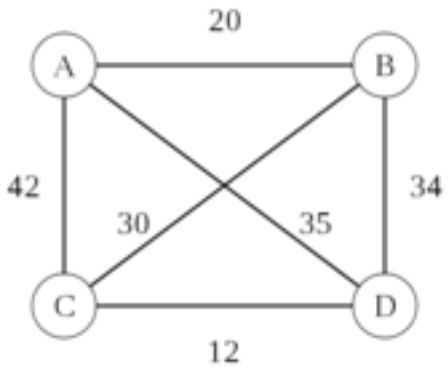

Figure 1:

- Trouver le plus petit circuit qui visite tous les noeuds exactement une fois.
- input : graphe dont les arêtes ont des poids.
- output : circuit de poids minimum
- Graphe orienté ou non-orientés (ici non orienté)
- Solution triviale: tester toutes les permutations possibles, mais $O((n-1)!)$
- Peut-on mieux faire ? Algo. exact en $O(n^2 2^n)$ (Held et Karp'62)
- NP-difficile (Karp'72) : réduction depuis Cycle Hamiltonien
- Pire encore: inapproximable ! (Orponen, Mannila'87 src: wikip)

Cas particuliers

TSP métrique

- On autorise plusieurs passage sur une ville
- Équivalent: graphe complet et poids satisfaisant l'inégalité triangulaire
- 1.5-approximable (Christophides'76)
- 2-approximable (plus simple)
  - Calculer un arbre couvrant de poids minimum (Kruskal ou Prim)
  - Faire un parcours en profondeur dans l'arbre
  - Enlever les répétitions
  - C'est 2-approx car $\leq 2MST$ et $MST \leq OPT$ (sinon pas MST)

TSP euclidien

- Cas particulier de métrique, ici coûts = distance euclidienne
- Approximable aussi près qu'on veut (PTAS) (Arora'98)
- Mais pas utilisable en pratique

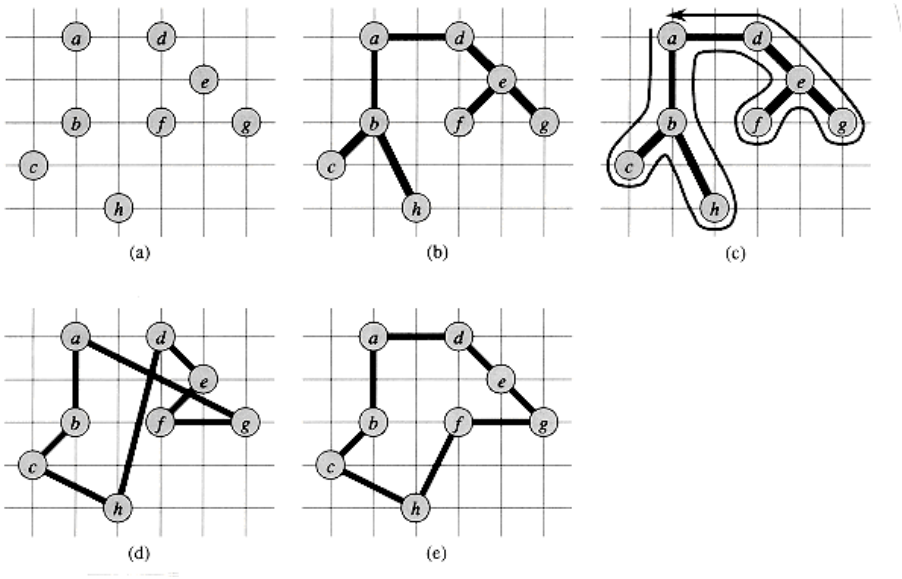

Figure 2: 2-approx

Heuristiques

En pratique, on utilise souvent des heuristiques simples

Nearest neighbor

- Choisir un noeuds initial arbitraire
- Ajouter le noeud non-visité le plus proche
- Recommencer tant qu'il y a des noeuds non visités

Random insertion

- Sélectionner un noeud non visité aléatoirement
- L'insérer à l'endroit du circuit qui augmente le moins le coût total
- Recommencer jusqu'à avoir tous les noeuds

Annexe : Arbres couvrants de poids minimaux

Algorithme de Kruskal

L'algorithme construit un arbre couvrant minimum en sélectionnant des arêtes par poids croissant. Plus précisément, l'algorithme considère toutes les arêtes du graphe par poids croissant (en pratique, on trie d'abord les arêtes du graphe par poids croissant) et pour chacune d'elle, il la sélectionne si elle ne crée pas un cycle. Complexité en $O(m \log m)$.

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty and F is not yet spanning
- remove an edge with minimum weight from S
- if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Algorithme de Prim

Faire croître un arbre depuis un sommet. On commence avec un seul sommet puis à chaque étape, on ajoute une arête de poids minimum adjacente à l'arbre en construction. Complexité en $O(nm)$.

The algorithm may informally be described as performing the following steps: - Initialize a tree with a single vertex, chosen arbitrarily from the graph. - Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree. - Repeat step 2 (until all vertices are in the tree).

Détection de cycle:

Faire un DFS depuis n'importe quel sommet. À tout moment, si l'un des voisins a déjà été atteint, alors il y a un cycle.

Autres

Christofides

- Make a Minimum Spanning Tree T
- Set $W = \{v | v \text{ has odd degree in tree } T\}$
- Compute a minimum weight matching M in the graph G[W].
- Look at the graph T+M. (Note: Eulerian!)
- Compute an Euler tour C' in T+M.
- Add shortcuts to C' to get a TSP-tour

Random insertion: randomly select a vertex

Each time: insert vertex at position that gives minimum increase of tour length -[→] Bonnes performances en pratique ! Voir aussi le fichier `{an-tsp.ppt}`.

Programmation dynamique

Par exemple pour TSP, cette technique permet de casser la barrière triviale de $O^*(n!)$ en réduisant la complexité à $O^*(2^n)$. Cela correspond au fait de calculer la solution pour chaque sous-ensemble possible de villes. En fait, on calcule plusieurs solutions pour chaque sous-ensemble : une pour chaque ville terminale possible. L'ajout d'une nouvelle ville à un ensemble se fait en considérant le minimum (parmi les solutions pour chaque ville terminale possible) de la longueur du chemin actuel + la jonction de la ville terminale à la nouvelle ville terminale.

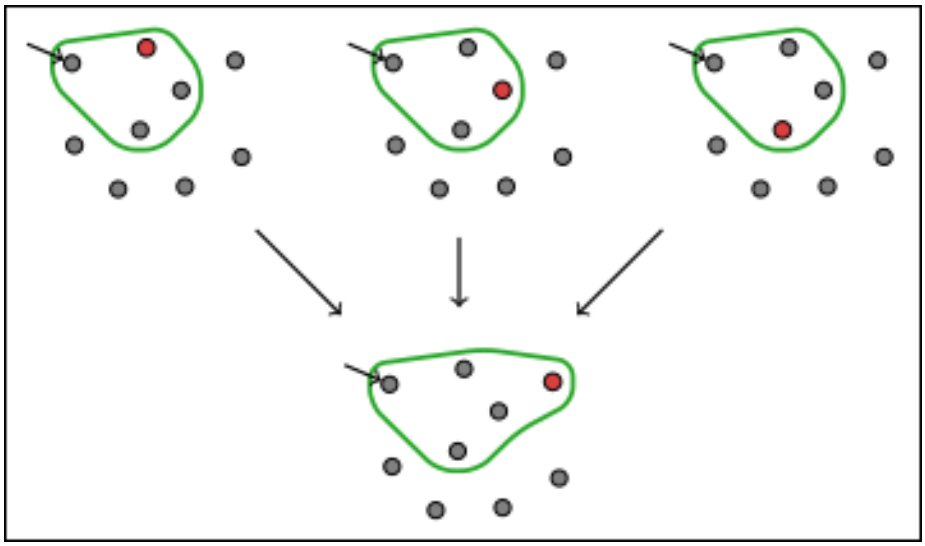

Figure 3: Prog dynamique pour le TSP