Algorithmes d'approximation

Montaigne 2023-2024

– mpi23@arrtes.net –

Généralités

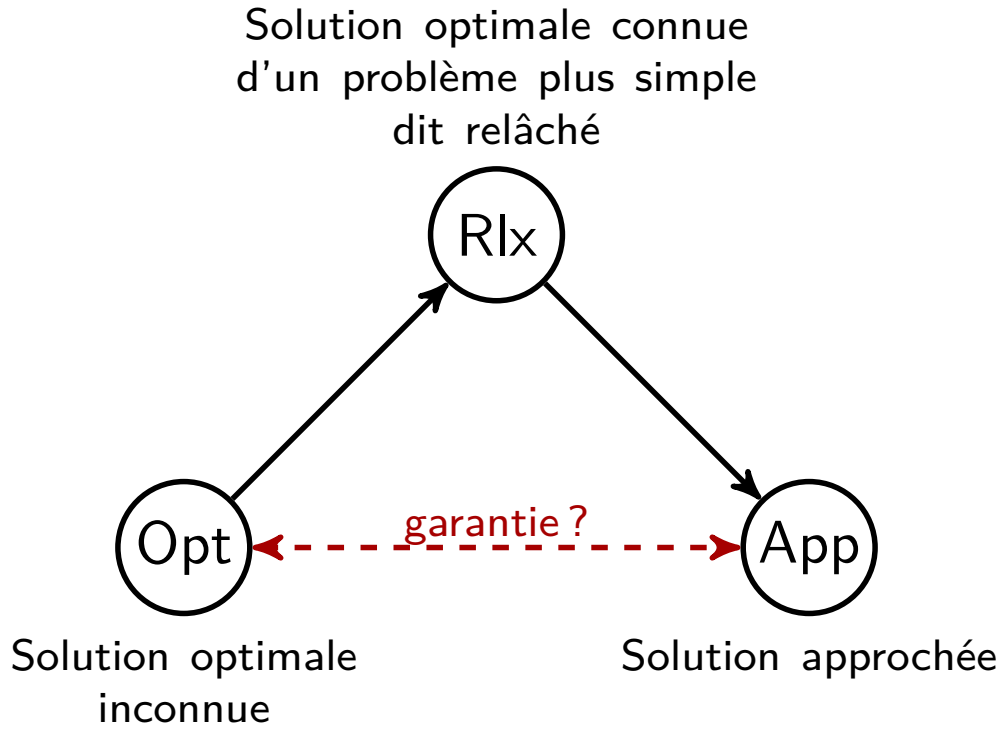
De nombreux problèmes NP-complets sont d'un intérêt pratique important. Leur complexité algorithmique semble cependant rendre illusoire de trouver une solution pour de nombreuses instances de leurs données d'entrée.

Toutefois, tout espoir n'est pas perdu si on se place dans deux situations particulières.

- Si ces algorithmes sont mis en œuvre sur de petite volumes de données, on peut raisonnablement espérer obtenir une solution, même avec un temps d'excécution exponentiel.
- On peut aussi trouver des solutions *quasi-optimales* à défaut d'être optimales, en temps polynomial. Un algorithme qui renvoie une telle solution approchée est appelé **algorithme d'approximation**.

Tout l'*art* de l'approximation est d'assurer qu'à chaque étape des calculs, on ne s'éloigne pas trop de l'*optimum* original.

Dans la mesure du possible, il s'agit donc d'obtenir des **garanties** sur les performances de l'algorithmes d'approximation.



De manière générale, on nomme **heuristique** tout algorithme supposé efficace en pratique qui produit un résultat dans garantie de qualité par rapport à la solution optimale.

Un **algorithme d'approximation** est une heuristique pour laquelle on garantit un certain niveau de performance chiffré.

Pour formaliser ces idées, notons Π un **problème d'optimisation** et I une **instance** de Π.

- $\textsc{Opt}(I)$: valeur de la solution optimale pour l'instance I.
- $A(I)$: valeur de la solution produite par l'algorithme A sur I.

Définition 1 (α-approximation)

Une **α-approximation** pour Π est un **algorithme polynomial** A qui, pour toute instance I de Π, renvoie une solution telle que :

- $A(I) \leqslant \alpha \times \textsc{Opt}(I)$ dans le cas d'une minimisation ;
- $A(I) \geqslant \alpha \times \textsc{Opt}(I)$ dans le cas d'une maximisation.

α est le **facteur d'approximation** de l'algorithme A.

Pour une *minimisation*, $\alpha \geqslant 1$ car $\textsc{Opt}(I) \leqslant A(I) \leqslant \alpha \times \textsc{Opt}(I)$.

Pour une *maximisation*, $\alpha \leqslant 1$ car $\alpha \times \textsc{Opt}(I) \leqslant A(I) \leqslant \textsc{Opt}(I)$.

Une 1-**approximation** est un **algorithme polynomial exact**.

Se méfier des terminologies et des définitions variables. Le contexte permet de ne pas se tromper.

Problème de la couverture par sommets

Couverture par sommets approchée

Algorithme 1 : VertexCoverApprox

Entrée : graphe $G = (V, E)$

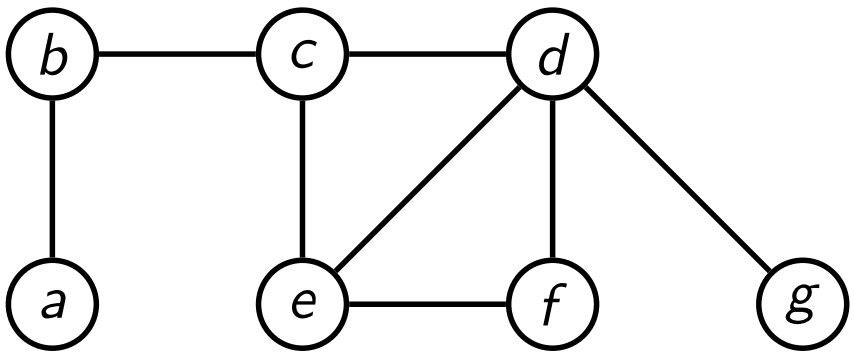
Sortie : couverture par sommets approchée C

```
1 C ← ∅
2 E' ← E
3 tant que E' ≠ ∅ faire
4     choisir une arête {u, v} de E'
5     C ← C ∪ {u, v}
6     supprimer de E' toutes les arêtes incidentes à u ou à v
7 renvoyer C
```

Mise en œuvre

$$V = \{a, b, c, d, e, f, g\}$$

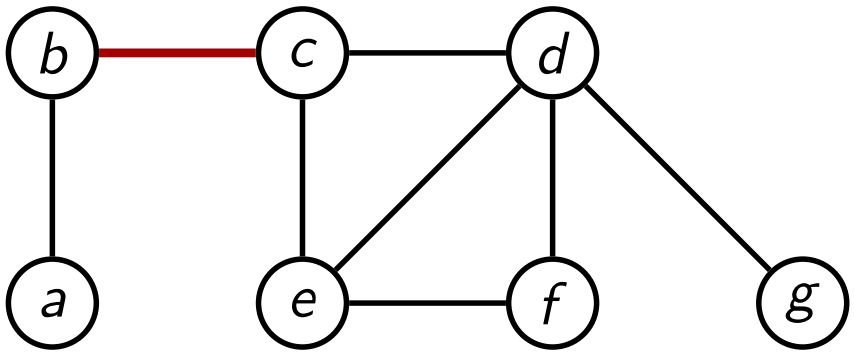
$$E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$



$$E' = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \varnothing$$

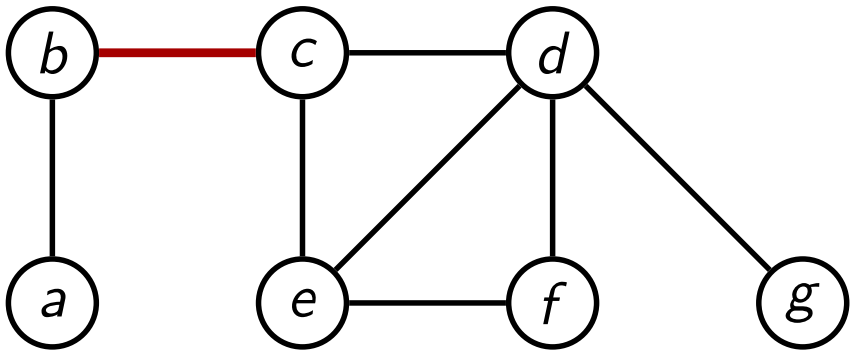
Mise en œuvre



$$E' = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \emptyset$$

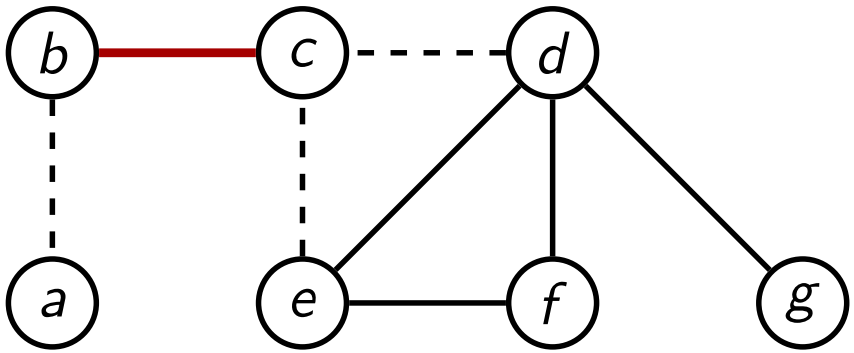
Mise en œuvre



$$E' = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c\}$$

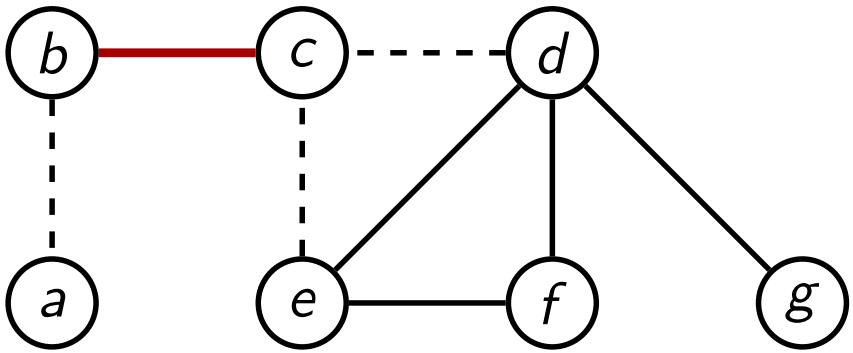
Mise en œuvre



$$E' = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c\}$$

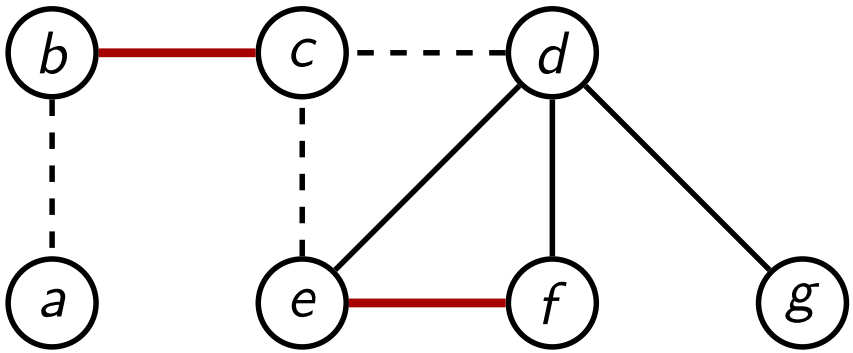
Mise en œuvre



$$E' = \{\{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c\}$$

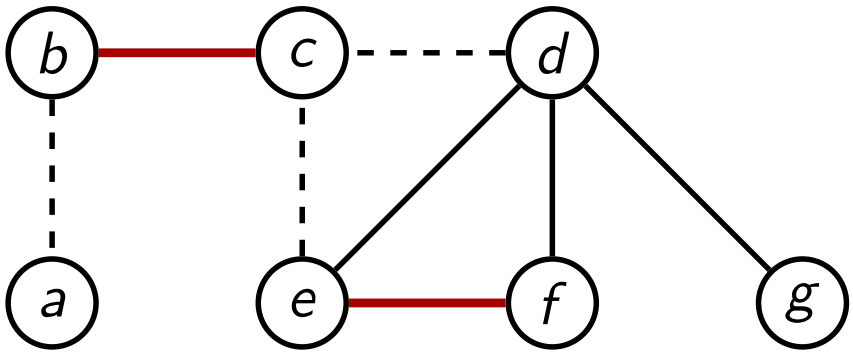
Mise en œuvre



$$E' = \{\{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c\}$$

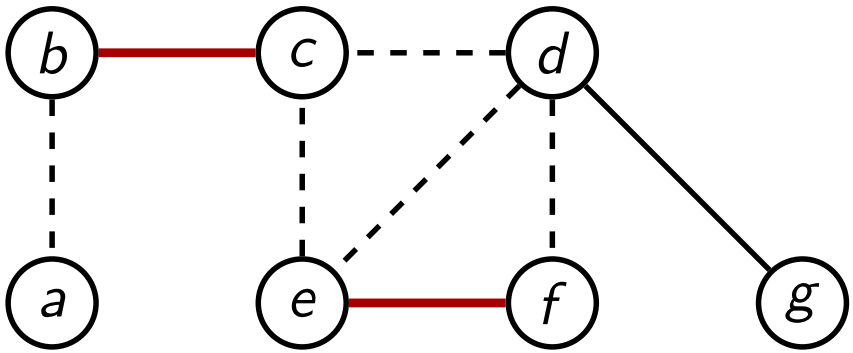
Mise en œuvre



$$E' = \{\{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c, e, f\}$$

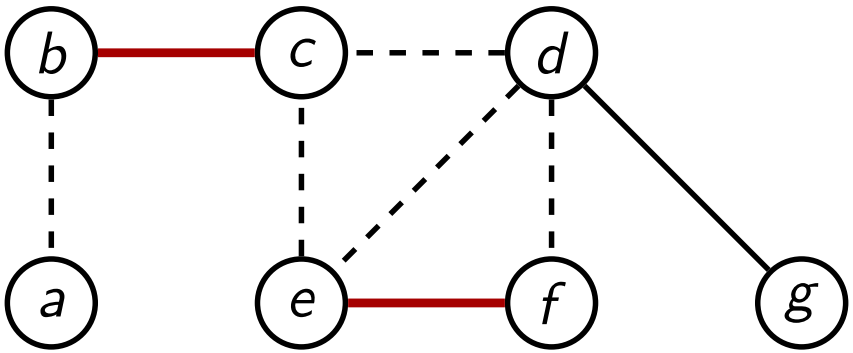
Mise en œuvre



$$E' = \{\{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}\}$$

$$C = \{b, c, e, f\}$$

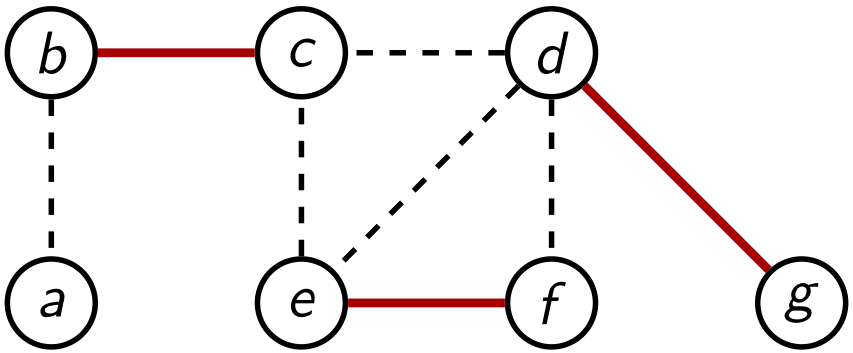
Mise en œuvre



$$E' = \{\{d, g\}\}$$

$$C = \{b, c, e, f\}$$

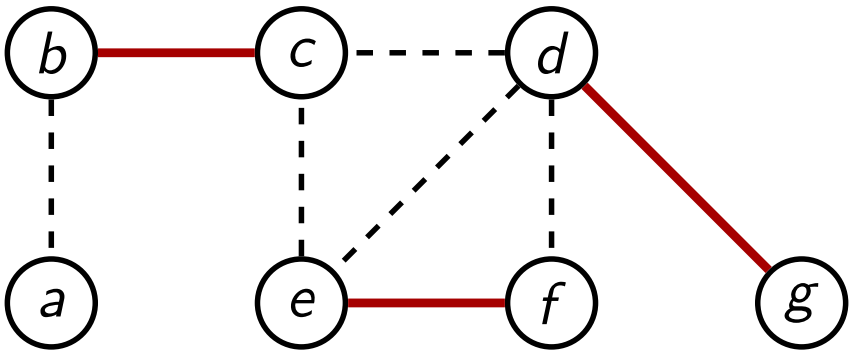
Mise en œuvre



$$E' = \{\{d, g\}\}$$

$$C = \{b, c, e, f\}$$

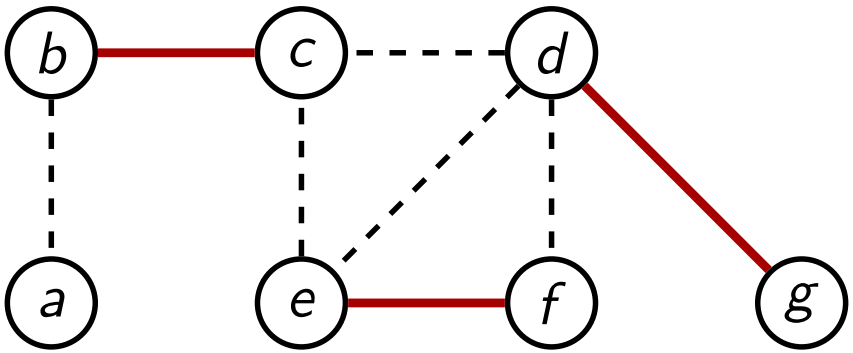
Mise en œuvre



$$E' = \{\{d, g\}\}$$

$$C = \{b, c, e, f, d, g\}$$

Mise en œuvre



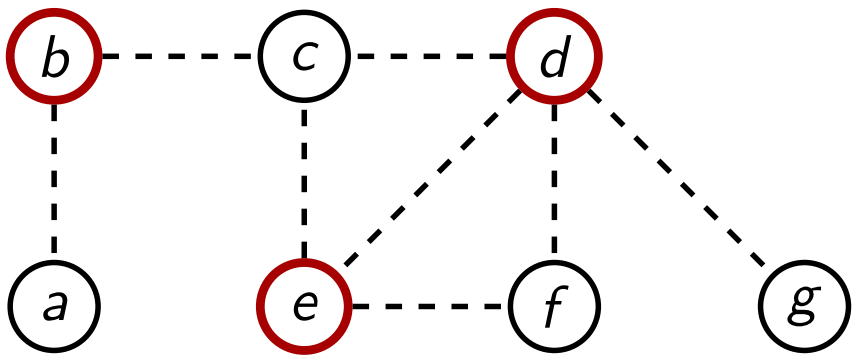
$$E' = \emptyset$$

$$C = \{b, c, e, f, d, g\}$$

Analyse

La mise en œuvre de l'algorithme renvoie ici une couverture des sommets définie par un ensemble C de taille 6.

D'autres choix mènent à des couvertures de tailles comprises en 3, taille optimale dont une couverture est donnée ci-dessous, et 6 puisque l'algorithme est une 2-approximation.



$C = \{b, d, e\}$

Complexité

La complexité de l'algorithme dépend essentiellement de l'efficacité des opérations d'extraction d'une arête (ligne 4) et de suppression des arêtes incidentes (lignes 6).

Une structure de données peut réaliser ces opérations en $O(1)$ de sorte que la **complexité** globale de l'algorithme est $O(|E|)$, soit **polynomiale** en la taille du graphe.

Garantie de performances

Théorème 2

VertexCoverApprox *est une* 2*-approximation du problème de la couverture par sommets.*

Démonstration

Désignons par E_c l'ensemble des arêtes choisies par l'algorithme à la ligne 4. Si une arête $\{u, v\}$ est choisie dans E', la ligne 6 supprime cette arête de E' ainsi que toutes celles contenant u et v. Les sommets u et v sont ajoutés à C à la ligne 5. Comme, par construction, deux arêtes de E_c ne peuvent pas avoir de sommet commun, à la fin de l'exécution de l'algorithme, C contient tous les sommets des arêtes de E_c et son cardinal est : $|C| = 2 \times |E_c|$.
Désignons à présent par C^* une couverture optimale de G. C^* est également une couverture des arêtes de E_c. Chacun de ses sommets est incident à au plus une arête de E_c. Par construction de E_c, C^* ne peut pas contenir moins qu'un des sommets de chaque arête de E_c afin de couvrir tous les sommets de G. Il suffit de raisonner par l'absurde pour s'en convaincre. Par conséquent : $|E_c| \leqslant |C^*|$.
Par conséquent :

$$|C| \leqslant 2 \times |C^*|$$

Problème du voyageur de commerce

Le problème

Un robot doit ramasser un ensemble d'objets en un minimum de temps et revenir au point de départ. L'ordre de ramassage n'a pas d'importance, seul le temps (ou la distance parcourue) doit être optimisé.

Une autre instance du même problème est celui où un hélicoptère doit inspecter un ensemble de plateformes offshore et revenir à son point de départ sur la côte. Il veut parcourir les plateformes en utilisant le moins de carburant possible. Une autre formulation est que l'hélicoptère possède une quantité de carburant C et il veut savoir s'il va pouvoir visiter toutes les plateformes avant de revenir.

La première formulation est un *problème d'optimisation* (la réponse est une valeur), alors que la seconde (avec un budget maximum C donné) est un *problème de décision* (la réponse est *oui* ou *non*).

Le problème

Dans la littérature et historiquement, on parle plutôt du problème du **voyageur de commerce**, ou *TSP* en anglais pour *Traveler Salesman Problem*. Pour effectuer une tournée comprenant un certain nombre de villes, un commercial doit déterminer l'ordre de visite qui minimise la longueur de sa tournée.

Problème du voyageur de commerce

Soit V un ensemble de points et d une distance sur V.
TSP consiste à trouver une tournée de longueur minimum passant exactement une seule fois par tous les points de V.
Autrement dit, il s'agit de trouver un ordre $v_0, v_1, \ldots, v_{n-1}$ sur les points de V de sorte que :

$$\sum_{i=0}^{n-1} d\left(v_i, v_{i+1 \bmod n}\right)$$

soit minimum.

Le problème

Il existe plusieurs variantes de ce problème. Dans la suite de l'exposé, on s'intéresse au **TSP métrique**, à savoir le TSP pour lequel d est une distance, fonction qui verifié notamment l'*inégalité triangulaire*.

$$d(A, B) \leqslant d(A, C) + d(C, B)$$

En ajoutant cette contrainte, il n'est alors plus nécessaire de préciser que la tournée doit passer une seule fois par chacun des points de V.

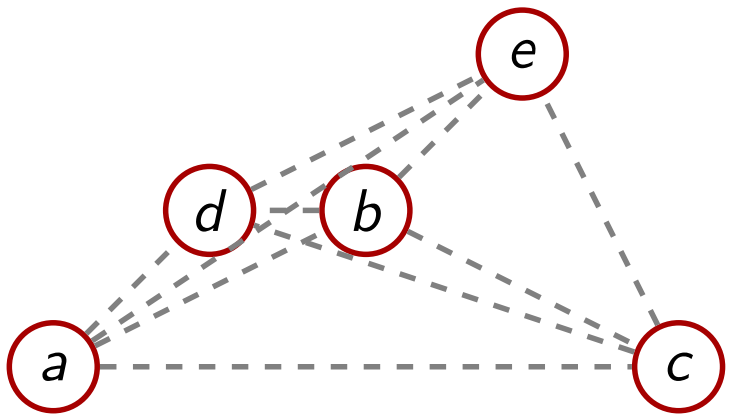
Le problème

Une autre variante est le *TSP non métrique* pour laquelle l'inégalité triangulaire n'est plus vérifiée. Par exemple, si les « distances » représentent des durées de voyage, aller de Bordeaux à Lyon en passant par Paris prend moins de temps qu'un trajet direct passant par le Massif Central !

$$d(\text{Bordeaux}, \text{Paris}) + d(\text{Paris}, \text{Lyon}) < d(\text{Bordeaux}, \text{Lyon})$$

Ajoutons l'existence d'une variante *asymétrique* illustrée par la différence des débits montant et descendant sur les réseaux informatiques.

Illustration

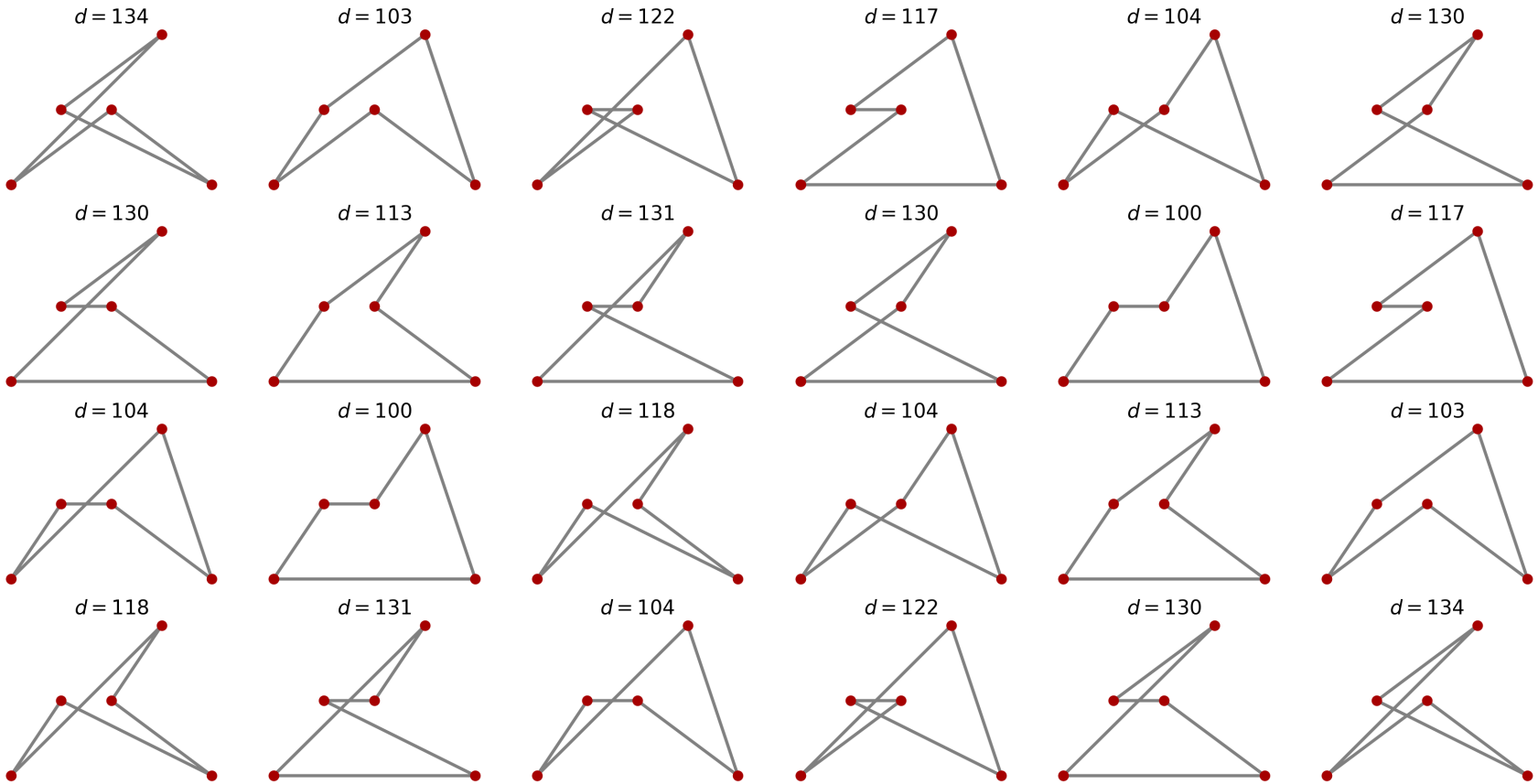


$$V = \{a, b, c, d, e\}$$

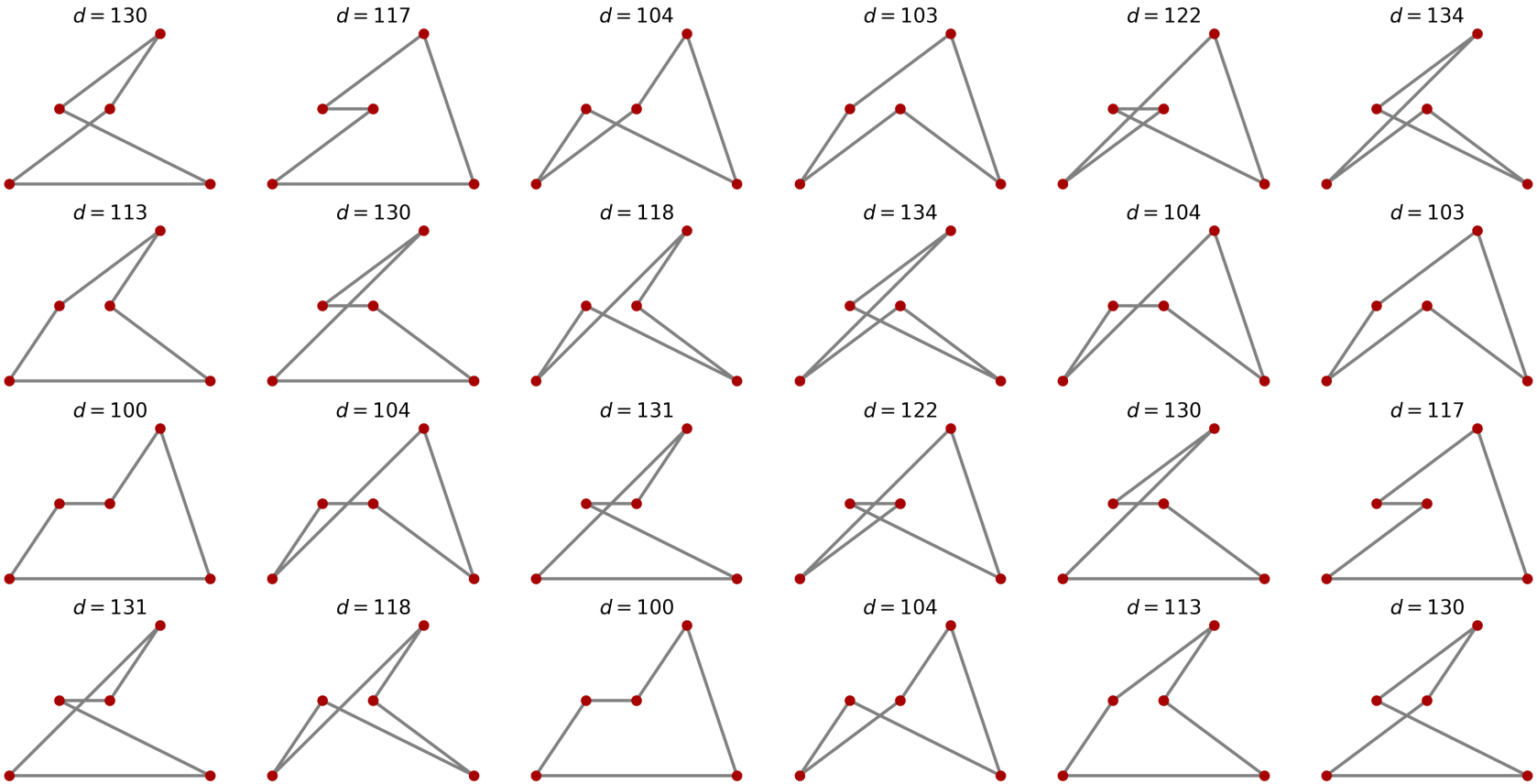
Tableau des distances d

	b	c	d	e
a	20	40	14	36
b		22	10	14
c			32	22
d				22

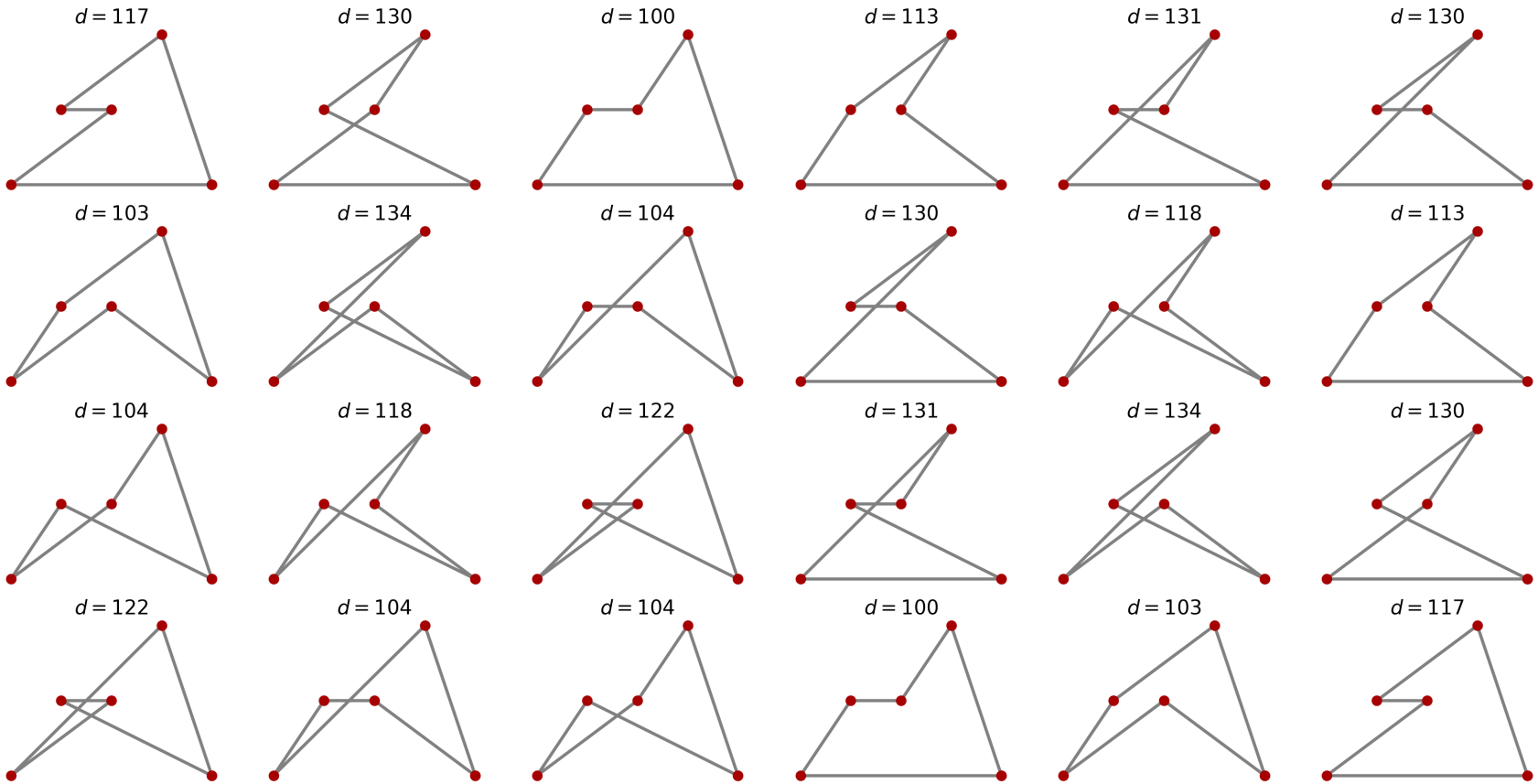
Recherche exhaustive



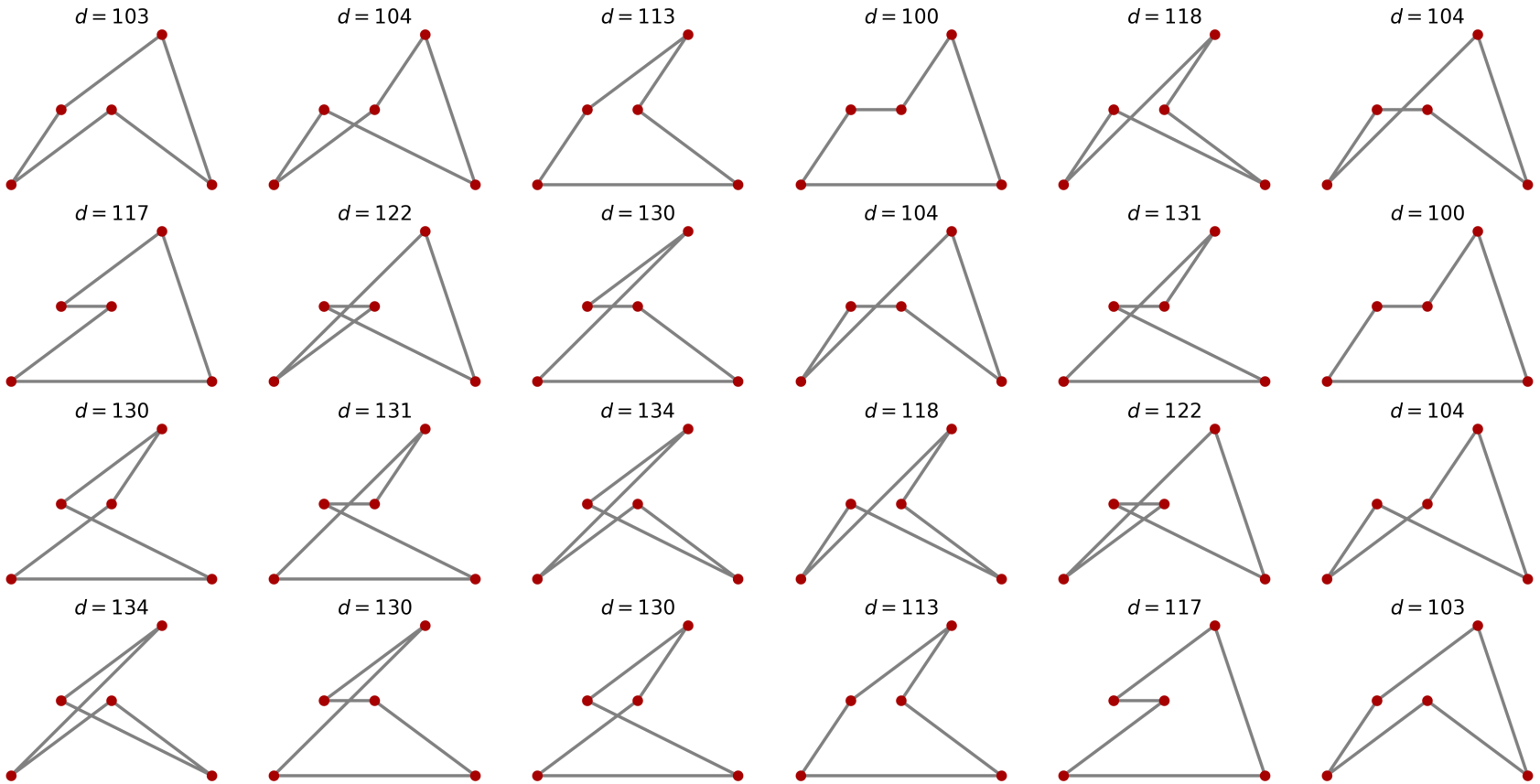
Recherche exhaustive



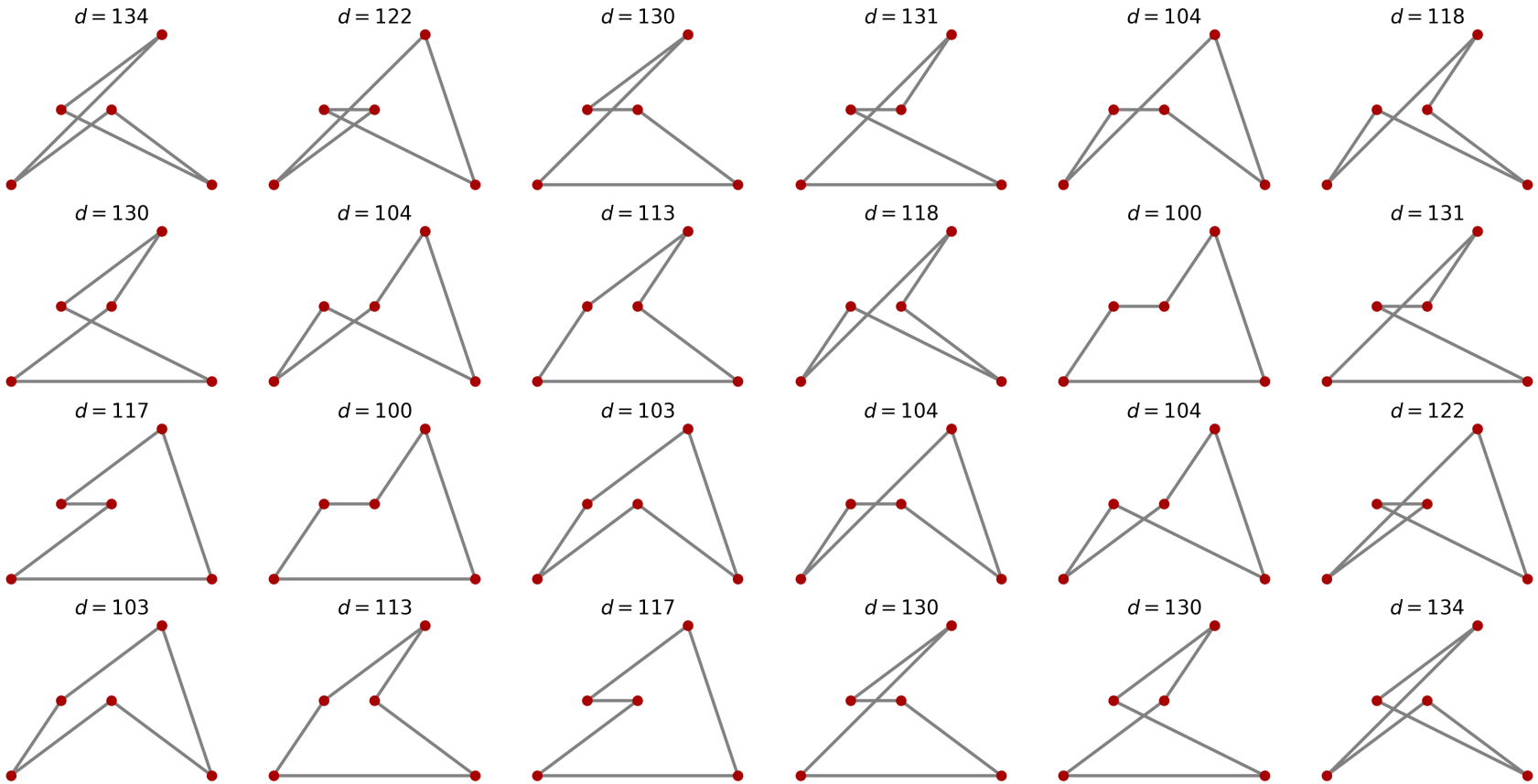
Recherche exhaustive



Recherche exhaustive



Recherche exhaustive



Problème TSP métrique approché

Algorithme 2 : TSPApprox

Entrée : une instance (V, d) du voyageur de commerce

Sortie : une tournée, *ie* un ordre sur les points de V

1 calculer un arbre couvrant minimum T sur le graphe complet défini par V et les arêtes valuées d

2 construire une tournée en suivant l'ordre des sommets lors d'un parcours DFS de T

Problème TSP métrique approché

Construction d'un arbre couvrant minimum (algorithme de Kruskal)

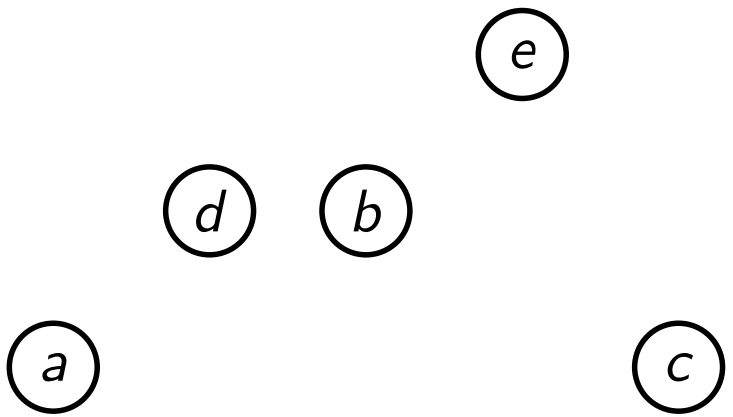


Tableau des distances d

	b	c	d	e
a	20	40	14	36
b		22	10	14
c			32	22
d				22

Problème TSP métrique approché

Construction d'un arbre couvrant minimum (algorithme de Kruskal)

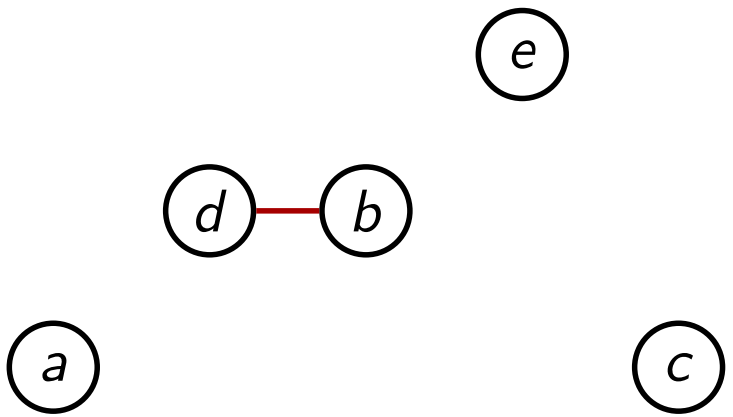


Tableau des distances d

	b	c	d	e
a	20	40	14	36
b		22	**10**	14
c			32	22
d				22

Problème TSP métrique approché

Construction d'un arbre couvrant minimum (algorithme de Kruskal)

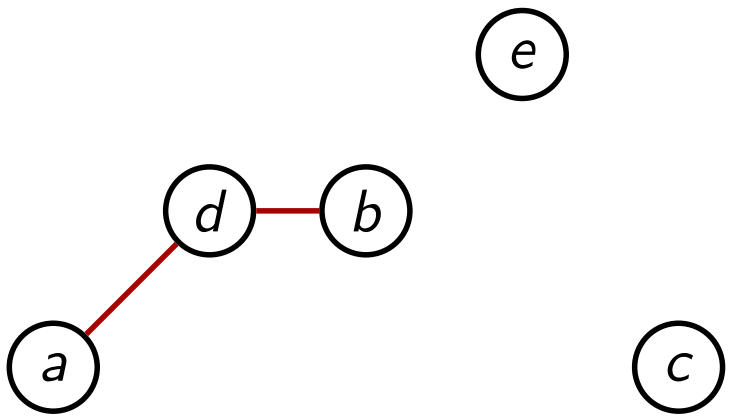


Tableau des distances d

	b	c	d	e
a	20	40	**14**	36
b		22	**10**	14
c			32	22
d				22

Problème TSP métrique approché

Construction d'un arbre couvrant minimum (algorithme de Kruskal)

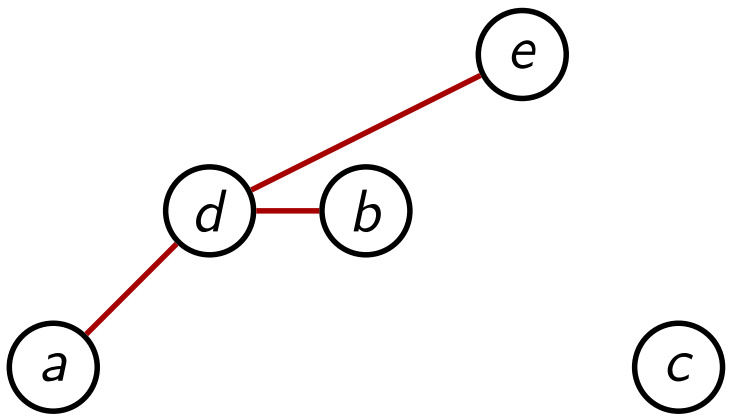


Tableau des distances d

	b	c	d	e
a	20	40	**14**	36
b		22	**10**	**14**
c			32	22
d				22

Problème TSP métrique approché

Construction d'un arbre couvrant minimum (algorithme de Kruskal)

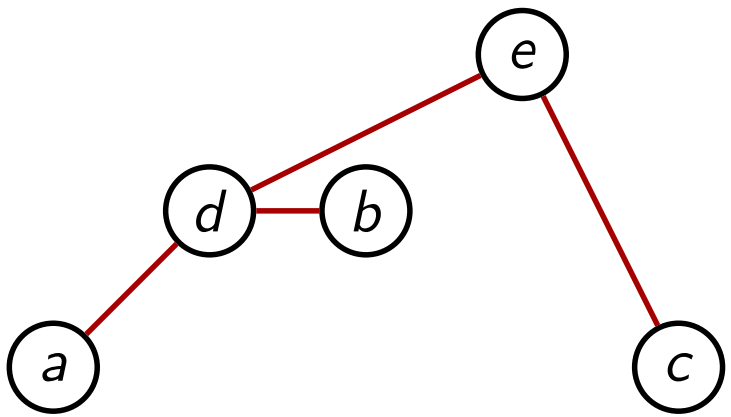
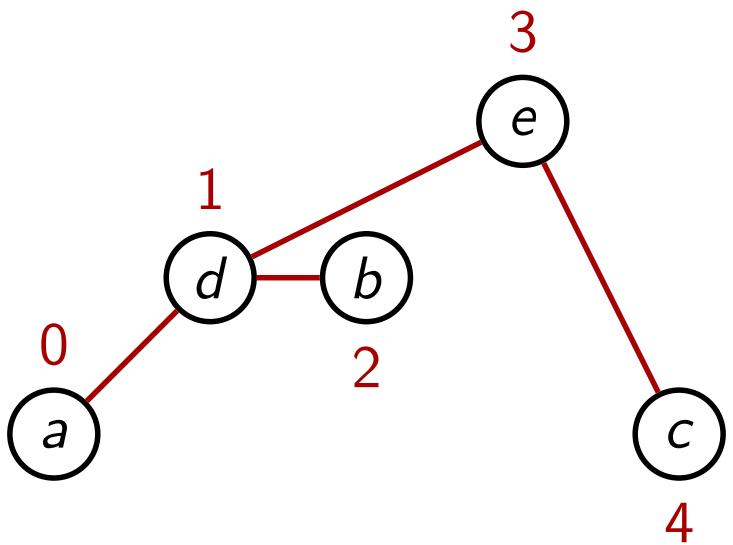


Tableau des distances d

	b	c	d	e
a	20	40	**14**	36
b		22	**10**	**14**
c			32	**22**
d				22

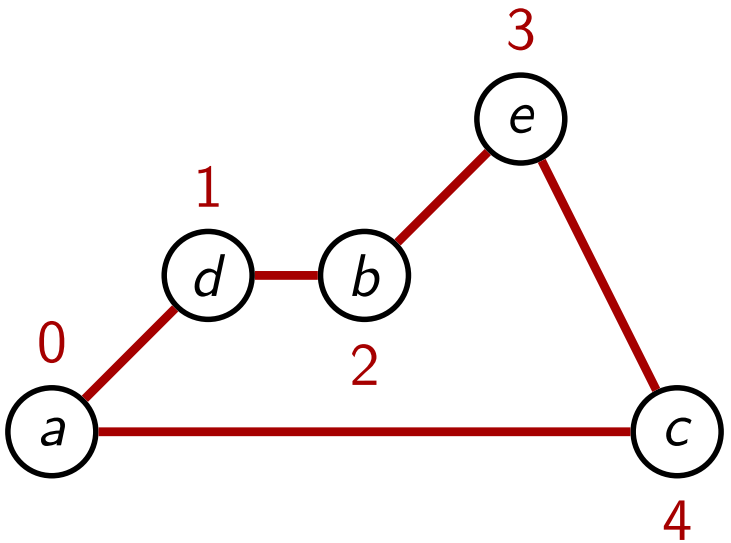
Problème TSP métrique approché

Parcours en profondeur depuis a et numérotation des sommets.



Problème TSP métrique approché

Construction de la tournée correspondante.



Distance de la tournée : $d = 100$.
C'est la plus petite distance pour ce problème.
D'autres tournées sont possibles. Trouvez les.

Complexité

L'algorithme de Kruskal calcule un arbre couvrant de poids minimum avec une complexité $O(m \log n)$ pour un graphe d'ordre n ayant m arêtes. Comme le graphe envisagé est complet, $m = \Theta(n^2)$ de sorte que la complexité est $O(n^2 \log n)$.
L'algorithme de Prim calculerait cet arbre avec une complexité $O(n^2)$, en utilisant les bonnes structures de données.

Le parcours en profondeur a un coût linéaire en le nombre de sommets n et d'arêtes $n-1$ du graphe, soit un coût global $O(n)$.

L'**algorithme approché** est donc de **complexité** $O(n^2 \log n)$.

Garantie de performance

Théorème 3

TSPApprox *est une 2-approximation de TSP.*

Pour établir ce résultat, on raisonne en deux temps.

- On montre tout d'abord que la longueur $\mathrm{Opt}(V, d)$ pour une instance (V, d) de TSP est plus grande que le poids de l'arbre T calculé à l'étape 1 de l'algorithme.

$$d(T) < \mathrm{Opt}(V, d)$$

- On montre ensuite que la tournée calculée à l'étape 2 après le parcours en profondeur de T est de longueur au plus $2 \times d(T)$.

$$d(\mathrm{TSPApprox}(V, d)) \leqslant 2 \times d(T)$$

Garantie de performance

Démonstration

T étant un arbre couvrant minimum, (T, d) est un graphe arête-valué de poids noté $d(T)$, somme des poids de chacune de ses arêtes.
Soit C une tournée quelconque (cycle simple) de (V, d), de poids $d(C)$. Si e désigne une arête de C, $C \setminus \{e\}$ est un arbre couvrant mais pas nécessairement de poids minimum. Par conséquent :

$$d(T) \leqslant d(C \setminus \{e\})$$

Comme $d(C \setminus \{e\}) = d(C) - d(\{e\})$, il vient :

$$d(T) < d(C)$$

En appliquant ce résultat au cycle de la tournée optimale $\textsc{Opt}(V, d)$, on obtient :

$$d(T) < \textsc{Opt}(V, d)$$

Garantie de performance

Démonstration

À présent, désignons par $(v_0, v_1, \ldots, v_{n-1}, v_0)$ la suite des points de la tournée calculée par l'algorithme d'approximation $\mathrm{TSPApprox}$ et par P_i le chemin, dans T, entre v_i et $v_{i+1 \bmod n}$. Par l'inégalité triangulaire, le poids du chemin P_i vérifie :

$$d(v_i, v_{i+1 \bmod n}) \leqslant d(P_i)$$

Alors :

$$d(\mathrm{TSPApprox}(V, d)) = \sum_{i=0}^{n-1} d(v_i, v_{i+1 \bmod n}) \leqslant \sum_{i=0}^{n-1} d(P_i)$$

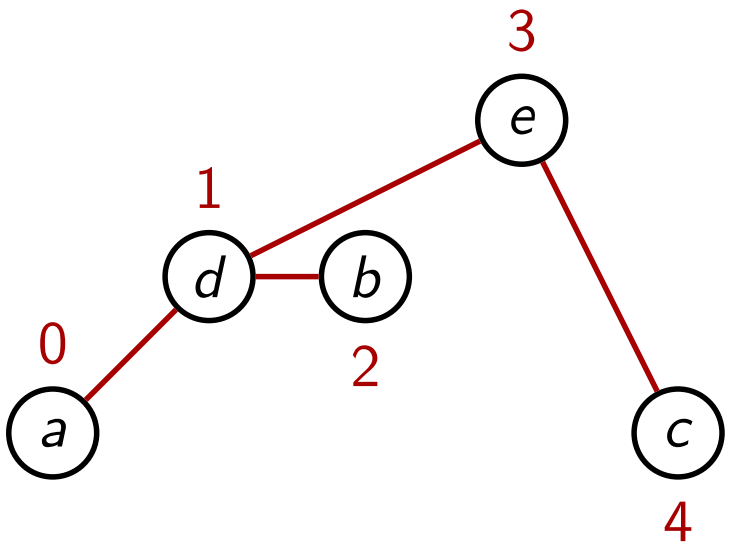
En suivant l'ordre des sommets rencontrés lors du parcours DFS de T, on construit un cycle qui visite deux fois chaque arête de V (faire un dessin pour s'en convaincre) dans un sens puis dans l'autre. À partir du sommet a, l'exemple précédent construit le cycle : $a \to d \to b \to d \to e \to c \to e \to d \to a$. Ainsi, chaque arête appartenant à deux chemins parmi $P_0, \ldots, P_{n-1}$, au plus et étant visitée *deux fois*, on a :

$$\sum_{i=0}^{n-1} d(P_i) = 2 \times d(T)$$

Finalement :

$$d(\mathrm{TSPApprox}(V, d)) \leqslant 2 \times d(T) < 2 \times d(\mathrm{Opt}(V, d)$$

Illustration du double parcours des arêtes



$v_0 = a$
$v_1 = d$
$v_2 = b$
$v_3 = e$
$v_4 = c$

Tournée calculée par $\mathrm{TSPAPPROX}$: $(v_0, v_1, v_2, v_3, v_4, v_0)$
Chemins dans T

$$P_0 = v_0 \to v_1 \qquad P_1 = v_1 \to v_2 \qquad P_2 = v_2 \to v_1 \to v_3$$

$$P_3 = v_3 \to v_4 \qquad P_4 = v_4 \to v_3 \to v_1 \to v_0$$

Arête $\{v_0, v_1\}$ dans P_0 et P_4 Arête $\{v_1, v_2\}$ dans P_1 et P_2
Arête $\{v_1, v_3\}$ dans P_2 et P_4 Arête $\{v_3, v_4\}$ dans P_3 et P_4