Algorithmique

Vlady Ravelomanana

IRIF – UMR CNRS 8243 - Paris 7
vlad@irif.fr

Master M1BIB — **NP-Complétude** —

Programme du cours

Le cours introduira les **méthodes générales** de **conception** et d'**analyse** d'**algorithmes** en commençant par les algorithmes de base :
tris classiques, structure de tas, arbres binaires de recherche, fonctions de hachage, graphes et parcours.
Ce cours présentera les principaux algorithmes utilisés pour résoudre de façon **approchée** ou **exacte** une partie des problèmes issus de la **biologie**. Le but est de permettre aux étudiants de comprendre les principes qui sous-tendent les méthodes implémentées dans les logiciels et de pouvoir se faire une opinion critique sur les outils disponibles.

Compétences visées

Cette connaissance devrait aider à éviter les deux écueils, la crainte des moyens informatiques ou leur passion excessive. Les **notions de complexité** (**NP-complétude** et **approximation polynomiale**) seront enseignées de façon “douce”, c'est-à-dire non abstraite, en s'appuyant sur les problèmes traités.

Introduction à la classe de complexité NP

En algorithmique, les **problèmes de décision** sont fondamentaux.

Exemple

Ecrire un algorithme décidant si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** **Oui** si le graphe contient un cycle et **Non** sinon.

Introduction à la classe de complexité NP

En algorithmique, les **problèmes de décision** sont fondamentaux.

Exemple

Ecrire un algorithme décidant si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** **Oui** si le graphe contient un cycle et **Non** sinon.

On a vu que pour le problème ci-dessus, il suffit d'**effeuiller** le graphe de manière récursive. Donc, ce problème peut se traiter en temps $O(n^2)$ (n étant le nombre de sommets). On dit alors que c'est un problème **traitable** (se traitant en temps **POLYNOMIAL**).

Introduction à la classe de complexité NP

En algorithmique, les **problèmes de décision** sont fondamentaux.

Exemple

Ecrire un algorithme décidant si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** **Oui** si le graphe contient un cycle et **Non** sinon.

On a vu que pour le problème ci-dessus, il suffit d'**effeuiller** le graphe de manière récursive. Donc, ce problème peut se traiter en temps $O(n^2)$ (n étant le nombre de sommets). On dit alors que c'est un problème **traitable** (se traitant en temps **POLYNOMIAL**). Un autre exemple de problème de décision est le suivant :

Exemple

Ecrire un algorithme décidant si un graphe contient un ensemble indépendant de taille k (pour k arbitrairement grand).

- **INPUT :** graphe (donné sous forme matricielle) et un entier k
- **OUTPUT :** **Oui** si le graphe contient un ens. ind. de taille k et **Non** sinon.

Introduction à la classe de complexité NP

En algorithmique, les **problèmes de décision** sont fondamentaux.

Exemple

Ecrire un algorithme décidant si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT : Oui** si le graphe contient un cycle et **Non** sinon.

On a vu que pour le problème ci-dessus, il suffit d'**effeuiller** le graphe de manière récursive. Donc, ce problème peut se traiter en temps $O(n^2)$ (n étant le nombre de sommets). On dit alors que c'est un problème **traitable** (se traitant en temps **POLYNOMIAL**). Un autre exemple de problème de décision est le suivant :

Exemple

Ecrire un algorithme décidant si un graphe contient un ensemble indépendant de taille k (pour k arbitrairement grand).

- **INPUT :** graphe (donné sous forme matricielle) et un entier k
- **OUTPUT : Oui** si le graphe contient un ens. ind. de taille k et **Non** sinon.

A ce jour, on ne connaît pas d'algorithme s'exécutant en temps **POLYNOMIAL** pour ce problème. On parle alors de **problèmes difficiles** algorithmiquement.

La classe NP (suite)

Pour un grand nombre de problèmes (ensemble indépendant, coloriage des graphes, problèmes d'emplois du temps, factorisation des nombres entiers), trouver une solution est **très difficile** mais il s'avère que si on vous propose une solution il est facile de vérifier que
(**lire par là qu'avec un algorithme s'exécutant en temps polynomial** on peut vérifier que) la solution proposée convient.

La classe NP (suite)

Pour un grand nombre de problèmes (ensemble indépendant, coloriage des graphes, problèmes d'emplois du temps, factorisation des nombres entiers), trouver une solution est **très difficile** mais il s'avère que si on vous propose une solution il est facile de vérifier que
(**lire par là qu'avec un algorithme s'exécutant en temps polynomial** on peut vérifier que) la solution proposée convient.

Exemple

Le problème **FACTM**.

- **INPUT :** Deux entiers n et M
- **OUTPUT :** Existe-t-il un diviseur de n inférieur à M ?

Remarque

Pour les problèmes arithmétiques la taille des entrées est en $\log_2(n)$.

La classe NP (suite)

Pour un grand nombre de problèmes (ensemble indépendant, coloriage des graphes, problèmes d'emplois du temps, factorisation des nombres entiers), trouver une solution est **très difficile** mais il s'avère que si on vous propose une solution il est facile de vérifier que
(**lire par là qu'avec un algorithme s'exécutant en temps polynomial** on peut vérifier que) la solution proposée convient.

Exemple

Le problème **FACTM**.

- **INPUT :** Deux entiers n et M
- **OUTPUT :** Existe-t-il un diviseur de n inférieur à M ?

Remarque

Pour les problèmes arithmétiques la taille des entrées est en $\log_2(n)$.

Hélas, on ne sait pas encore résoudre **FACTM** en temps **POLYNOMIAL** de $\log_2(n)$. Exemple : $2\,147\,483\,647$ est un nombre premier donc la réponse au problème de décision est **Non**. Mais si on nous propose une réponse comme par exemple 2 699, dans **FACTM(**$2\,147\,483\,641, 2\,800$**)** on vérifie bien plus facilement qu'on a

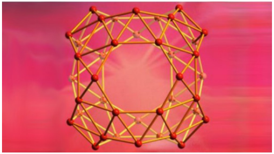
$$2\,147\,483\,641/2\,699 = 795\,659\,.$$

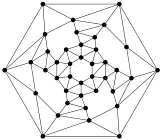
La classe NP (suite)

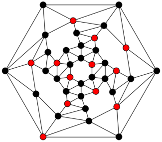
Les problèmes de décision suivants sont difficiles (donc à ce jour on ne connaît pas de solutions algorithmiques polynomiales) :

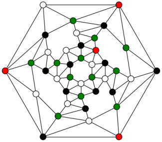
- décider si un graphe est k-colorable
- décider si un graphe contient un cycle de longueur k
- décider si un graphe contient un ensemble indépendant (ou une clique) de taille k
- décider si un graphe est couvert par k sommets
- $\cdots$

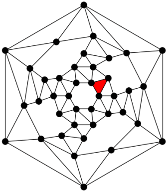
La classe NP : le borospherene [1]

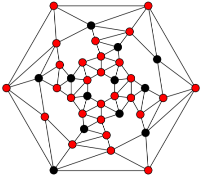












Définitions non-formelles

Les deux principales classes de complexité :

- La **classe** *P* est celle des problèmes pour lesquels **il existe un algorithme polynomial en la taille des données** permettant de les résoudre.
- La **classe** *NP* est celle des problèmes pour lesquels **on ne connaît pas d'algorithmes polynomiaux pour les résoudre** mais on peut cependant **vérifier** en un temps polynomial si une solution proposée convient.

Définitions non-formelles

Les deux principales classes de complexité :

- La **classe** P est celle des problèmes pour lesquels **il existe un algorithme polynomial en la taille des données** permettant de les résoudre.

- La **classe** NP est celle des problèmes pour lesquels **on ne connaît pas d'algorithmes polynomiaux pour les résoudre** mais on peut cependant **vérifier** en un temps polynomial si une solution proposée convient.

Remarque

Le problème de savoir si $P = NP$ est un des problèmes scientifiques de notre millénaire et il a été mis à prix à 1 000 000$.

Problèmes NP-complets

Un problème P est polynomialement réductible à un autre problème Q s'il existe un algorithme polynomial permettant de ramener la recherche d'une solution de P à celle d'une solution de Q.

Problèmes NP-complets

Un problème P est polynomialement réductible à un autre problème Q s'il existe un algorithme polynomial permettant de ramener la recherche d'une solution de P à celle d'une solution de Q.

Le problème **SAT** (cf. tableau)

Problèmes NP-complets

Un problème P est polynomialement réductible à un autre problème Q s'il existe un algorithme polynomial permettant de ramener la recherche d'une solution de P à celle d'une solution de Q.

Le problème **SAT** (cf. tableau)

Un problème dans NP équivalent par réduction polynomiale au problème **SAT** (qui est dans NP) est dit NP-complets.

Problèmes NP-complets

Un problème P est polynomialement réductible à un autre problème Q s'il existe un algorithme polynomial permettant de ramener la recherche d'une solution de P à celle d'une solution de Q.

Le problème **SAT** (cf. tableau)

Un problème dans NP équivalent par réduction polynomiale au problème **SAT** (qui est dans NP) est dit NP-complets.

Pour montrer qu'un problème est NP-complet, il faut montrer donc qu'il est dans NP et montrer qu'il est équivalent à un problème NP-complet déjà connu.

Problèmes NP-complets

Un problème P est polynomialement réductible à un autre problème Q s'il existe un algorithme polynomial permettant de ramener la recherche d'une solution de P à celle d'une solution de Q.

Le problème **SAT** (cf. tableau)

Un problème dans NP équivalent par réduction polynomiale au problème **SAT** (qui est dans NP) est dit NP-complets.

Pour montrer qu'un problème est NP-complet, il faut montrer donc qu'il est dans NP et montrer qu'il est équivalent à un problème NP-complet déjà connu.

Remarque

Si un seul des problèmes NP-complets peut être résolu en temps polynomial alors tous le sont.

L'exemple typique du problème d'allocation de fréquences

Les données du problèmes sont les suivantes. On a un certain nombre de transmetteurs radios qui peuvent transmettre sur un ensemble de fréquences. Les transmetteurs qui sont géographiquement proches ne peuvent pas transmettre sur les mêmes fréquences (problème d'interférence radio). Le problème qui consiste à allouer aux transmetteurs un nombre minimum de fréquences sans qu'il y ait interférence est NP-complet.

La preuve est donnée par les 3 étapes suivantes.

L'exemple typique du problème d'allocation de fréquences

Les données du problèmes sont les suivantes. On a un certain nombre de transmetteurs radios qui peuvent transmettre sur un ensemble de fréquences. Les transmetteurs qui sont géographiquement proches ne peuvent pas transmettre sur les mêmes fréquences (problème d'interférence radio). Le problème qui consiste à allouer aux transmetteurs un nombre minimum de fréquences sans qu'il y ait interférence est NP-complet.

La preuve est donnée par les 3 étapes suivantes.
Modélisation. On modélise ce problème avec des **graphes** : les transmetteurs sont les sommets/nœuds et deux transmetteurs qui peuvent interferer partagent une arête.

L'exemple typique du problème d'allocation de fréquences

Les données du problèmes sont les suivantes. On a un certain nombre de transmetteurs radios qui peuvent transmettre sur un ensemble de fréquences. Les transmetteurs qui sont géographiquement proches ne peuvent pas transmettre sur les mêmes fréquences (problème d'interférence radio). Le problème qui consiste à allouer aux transmetteurs un nombre minimum de fréquences sans qu'il y ait interférence est NP-complet.

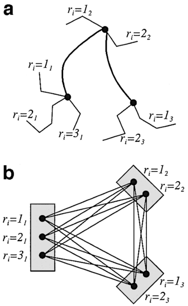
La preuve est donnée par les 3 étapes suivantes.
Modélisation. On modélise ce problème avec des **graphes** : les transmetteurs sont les sommets/nœuds et deux transmetteurs qui peuvent interferer partagent une arête.
Certificat vérifiable en temps polynomial. Si on nous donne une solution, on peut **vérifier en temps linéaire** (donc polynomial) en le nombre de transmetteurs que cette solution convient.

L'exemple typique du problème d'allocation de fréquences

Les données du problèmes sont les suivantes. On a un certain nombre de transmetteurs radios qui peuvent transmettre sur un ensemble de fréquences. Les transmetteurs qui sont géographiquement proches ne peuvent pas transmettre sur les mêmes fréquences (problème d'interférence radio). Le problème qui consiste à allouer aux transmetteurs un nombre minimum de fréquences sans qu'il y ait interférence est NP-complet.

La preuve est donnée par les 3 étapes suivantes.
Modélisation. On modélise ce problème avec des **graphes** : les transmetteurs sont les sommets/nœuds et deux transmetteurs qui peuvent interferer partagent une arête.
Certificat vérifiable en temps polynomial. Si on nous donne une solution, on peut **vérifier en temps linéaire** (donc polynomial) en le nombre de transmetteurs que cette solution convient.
Réduction au problème de coloration de graphes. Le problème de coloration de graphe consiste à décider si oui/non un graphe G est colorable en k-couleurs. Avec ce qui précède, le problème d'allocation de fréquences est clairement le même que le problème de coloration de graphe qui lui est (admis) NP-complet. On en conclut que le problème d'allocation de fréquences est aussi NP-complet.

Un exemple issu de la bio : “Protein Design is NP-hard”



(a) Representative protein backbone with three rotamers at position $i = 1$ and two rotamers at positions $i = 2$ and $i = 3$. **(b)** The corresponding graph. Each box represents a backbone position ; each circular node represents a rotamer alternative ; each edge connecting two nodes is weighted by the pairwise energy. In this picture, the

goal is to choose the one node within each box that minimizes the sum of the three edge weights connecting t

D'après **N. A. Pierce** et **E. Winfree** dans ***Protein Engineering, Design & Selection*** Vol. 15, Issue 10 pp. 779-782 (2002).

En algorithmique, les **problèmes de décision**, **d'optimisation** et **de recherche** sont centraux et ils peuvent être **très difficiles** (au sens très coûteux).

En algorithmique, les **problèmes de décision**, **d'optimisation** et **de recherche** sont centraux et ils peuvent être **très difficiles** (au sens très coûteux).

Exemple

Ecrire un algorithme **décidant** si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT : Oui** si le graphe contient un cycle et **Non** sinon.

En algorithmique, les **problèmes de décision**, **d'optimisation** et **de recherche** sont centraux et ils peuvent être **très difficiles** (au sens très coûteux).

Exemple

Ecrire un algorithme **décidant** si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** **Oui** si le graphe contient un cycle et **Non** sinon.

Exemple

Ecrire un algorithme **calculant** le plus grand cycle d'un graphe revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** le plus grand cycle du graphe.

En algorithmique, les **problèmes de décision**, **d'optimisation** et **de recherche** sont centraux et ils peuvent être **très difficiles** (au sens très coûteux).

Exemple

Ecrire un algorithme **décidant** si un graphe contient un cycle revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** **Oui** si le graphe contient un cycle et **Non** sinon.

Exemple

Ecrire un algorithme **calculant** le plus grand cycle d'un graphe revient à écrire un algorithme tel que :

- **INPUT :** graphe (donné sous forme matricielle)
- **OUTPUT :** le plus grand cycle du graphe.

Remarque

A ce jour, pour beaucoup de problèmes algorithmiques on ne connaît pas d'algorithme s'exécutant en temps **POLYNOMIAL** en la taille de l'entrée pour donner des solutions exactes, on s'oriente alors vers des **solutions approchées** de bonne qualitée pouvant se calculer en temps **POLYNOMIAL** :
on parle alors d'**approximation polynomiale de problèmes difficiles**.

Définition 1.

Un **algorithme d'approximation** a un **ratio d'approximation** $\rho(n)$ si pour toute entrée de taille n on a

$$\max\left\{\frac{C}{C^\star}, \frac{C^\star}{C}\right\} \leqslant \rho(n)$$

où C et $C^\star$ représentent les coûts respectifs de la **solution approchée** et de la **solution optimale**. Un tel algorithme d'approximation est alors appelé **algorithme ρ-approché**.

Définition 3.

Un **algorithme d'approximation** a un **ratio d'approximation** $\rho(n)$ si pour toute entrée de taille n on a

$$\max\left\{\frac{C}{C^\star}, \frac{C^\star}{C}\right\} \leqslant \rho(n)$$

où C et $C^\star$ représentent les coûts respectifs de la **solution approchée** et de la **solution optimale**. Un tel algorithme d'approximation est alors appelé **algorithme ρ-approché**.

Remarque

Il y aura souvent un *compromis* entre la complexité de l'algorithme d'approximation et la qualité de la solution qu'il fournira.

Définition 5.

Un **algorithme d'approximation** a un **ratio d'approximation** $\rho(n)$ si pour toute entrée de taille n on a

$$\max\left\{\frac{C}{C^\star}, \frac{C^\star}{C}\right\} \leqslant \rho(n)$$

où C et $C^\star$ représentent les coûts respectifs de la **solution approchée** et de la **solution optimale**. Un tel algorithme d'approximation est alors appelé **algorithme ρ-approché**.

Remarque

Il y aura souvent un *compromis* entre la complexité de l'algorithme d'approximation et la qualité de la solution qu'il fournira.

Définition 6.

Un **schéma d'approximation** (en abrégé **PTAS** pour *Polynomial-Time Approximation Scheme*) pour un problème d'optimisation est un algorithme d'approximation ayant pour entrées

- une instance du problème (de taille n) **et**
- un nombre $\varepsilon(n) > 0$

et retourne une **solution $(1+\varepsilon)$-approchée**.

Définition 7.

Un **algorithme d'approximation** a un **ratio d'approximation** $\rho(n)$ si pour toute entrée de taille n on a

$$\max\left\{\frac{C}{C^\star}, \frac{C^\star}{C}\right\} \leqslant \rho(n)$$

où C et $C^\star$ représentent les coûts respectifs de la **solution approchée** et de la **solution optimale**. Un tel algorithme d'approximation est alors appelé **algorithme ρ-approché**.

Remarque

Il y aura souvent un *compromis* entre la complexité de l'algorithme d'approximation et la qualité de la solution qu'il fournira.

Définition 8.

Un **schéma d'approximation** (en abrégé **PTAS** pour *Polynomial-Time Approximation Scheme*) pour un problème d'optimisation est un algorithme d'approximation ayant pour entrées

- une instance du problème (de taille n) **et**
- un nombre $\varepsilon(n) > 0$

et retourne une **solution $(1+\varepsilon)$-approchée**.

Remarque

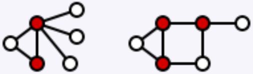
Souvent, on demande de plus que le temps d'exécution soit polynomial en n pour chaque ε fixé. Par exemple en $\mathcal{O}(n^{1/\varepsilon})$.

Problème de couverture par sommets (Vertex Cover)

Définition 9.

Une couverture par sommets, ou transversale d'un graphe G est un ensemble C de sommets tel que chaque arête de $G = (V, E)$ est incidente à au moins un sommet de C.

Exemple

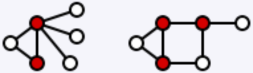


Problème de couverture par sommets (Vertex Cover)

Définition 11.

Une couverture par sommets, ou transversale d'un graphe G est un ensemble C de sommets tel que chaque arête de $G = (V, E)$ est incidente à au moins un sommet de C.

Exemple



Définition 12.

Le problème de **couverture minimum par sommets** (ou problème du **transversal minimum** ou *Vertex Cover*) consiste à trouver un ensemble minimum de sommets qui **couvre toutes les arêtes dans un graphe** :

- **INPUT :** graphe
- **OUTPUT :** le plus petit ensemble de sommets couvrant toutes les arêtes du graphe.

Un algo d'approximation pour Vertex Cover

Entrée : Un graphe $G = (V, E)$.
Sortie : Un ensemble de sommets couvrant.
$C := \emptyset$
$E' := E$
tant que $E' \neq \emptyset$ **faire**
  $C := C \cup \{u, v\}$ $(\star\ (u, v) \in E'\ \star)$
  $E' := E' - \{$toutes les arêtes de E'incidentes à u et à $v\}$
fin tant que
retourner C

Un algo d'approximation pour Vertex Cover

Entrée : Un graphe $G = (V, E)$.
Sortie : Un ensemble de sommets couvrant.
$C := \emptyset$
$E' := E$
tant que $E' \neq \emptyset$ **faire**
  $C := C \cup \{u, v\}$ $(\star\ (u, v) \in E'\ \star)$
  $E' := E' - \{$toutes les arêtes de E'incidentes à u et à $v\}$
fin tant que
retourner C

Théorème

L'algorithme ci-dessus est un **algorithme polynomial** 2-**approché**.

Un algo d'approximation pour Vertex Cover

Entrée : Un graphe $G = (V, E)$.
Sortie : Un ensemble de sommets couvrant.
$C := \emptyset$
$E' := E$
tant que $E' \neq \emptyset$ **faire**
  $C := C \cup \{u, v\}$ $(\star\ (u, v) \in E'\ \star)$
  $E' := E' - \{$toutes les arêtes de E'incidentes à u et à $v\}$
fin tant que
retourner C

Théorème

L'algorithme ci-dessus est un **algorithme polynomial** 2-**approché**.

Preuve

Soit C l'ensemble de sommets trouvé par l'algorithme et $C^\star$ l'optimal (le plus petit ensemble de sommets couvrant). Soit A l'ensemble des arêtes (u, v) "enlevées" par l'algorithme. On a

$$|C| = 2.|A|\,.$$

Puisque les arêtes de A sont indépendantes, on a $|A| \leqslant |C^\star$ et donc

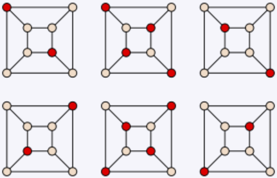
$$|C| \leqslant 2.|C^\star|\,.$$

Ensemble indépendant de taille maximale (Maximum Independent Set ou MaxIS)

Définition 13.

Instance : Un graphe $G = (V, E)$.
Problème : Le plus grand ensemble indépendant de sommets.

Exemple



Un algorithme d'approximation pour MaxIS

On définit $\delta = \frac{1}{n} \sum_{v \in V} \mathbf{deg}(v)$ et $N(v) =$ **les voisins de** v.

$S := \emptyset$
tant que $V(G) \neq \emptyset$ **faire**
  Trouver v de degré minimal.
  $S := S \cup \{v\}$
  $G := G - (v \cup N(v))$.
fin tant que
retourner S

Un algorithme d'approximation pour MaxIS

On définit $\delta = \frac{1}{n} \sum_{v \in V} \mathbf{deg}(v)$ et $N(v) =$ **les voisins de** v.

$S := \emptyset$
tant que $V(G) \neq \emptyset$ **faire**
  Trouver v de degré minimal.
  $S := S \cup \{v\}$
  $G := G - (v \cup N(v))$.
fin tant que
retourner S

Théorème

L'algorithme ci-dessus calcule un ensemble indépendant de taille $q \geqslant n/(\delta + 1)$.

Un algorithme d'approximation pour MaxIS

On définit $\delta = \frac{1}{n} \sum_{v \in V} \mathbf{deg}(v)$ et $N(v) =$ **les voisins de** v.

$S := \emptyset$
tant que $V(G) \neq \emptyset$ **faire**
  Trouver v de degré minimal.
  $S := S \cup \{v\}$
  $G := G - (v \cup N(v))$.
fin tant que
retourner S

Théorème

L'algorithme ci-dessus calcule un ensemble indépendant de taille $q \geqslant n/(\delta + 1)$.

Preuve

Soit v_i le sommet choisi à l'étape i et $d_i = \deg(v_i)$. A chaque étape, on supprime $d_i + 1$ sommets de degré au moins d_i chacun. Donc, on a $s_i \geqslant d_i(d_i + 1)$ et
$\sum_{i=1}^{q} s_i \geqslant \sum_{i=1}^{q} d_i(d_i + 1) \leqslant \sum_{v \in V} \mathbf{deg}(v) = n\delta$. On a donc

$$\delta n + n \geqslant \sum_{i=1}^{q} d_i(d_i + 1) + (d_i + 1) = \sum_{i=1}^{q} (d_i + 1)^2 \geqslant \frac{n^2}{q}$$

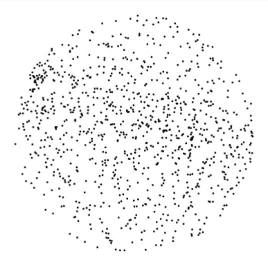
D'où le résultat.

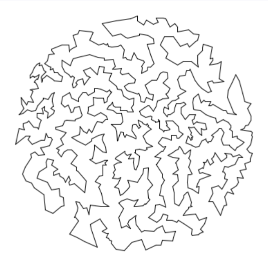
Le problème du voyageur de commerce (TSP : Traveling Salesman Problem)

Définition 14.

Instance : Un graphe complet $G = (V, E)$ et une fonction de poids des arêtes $c : E \to \mathbb{R}^+$.
Problème : Trouver un cycle passant par tous les sommets (cycle hamiltonien) de poids minimum.

Exemple





Un algorithme d'approximation pour TSP

On part d'un sommet $v \in V$.
On construit T : un arbre couvrant de poids minimum du graphe enraciné en v.
On parcours T en ordre "préfixe" et on renvoie le cycle associé.

Exemple