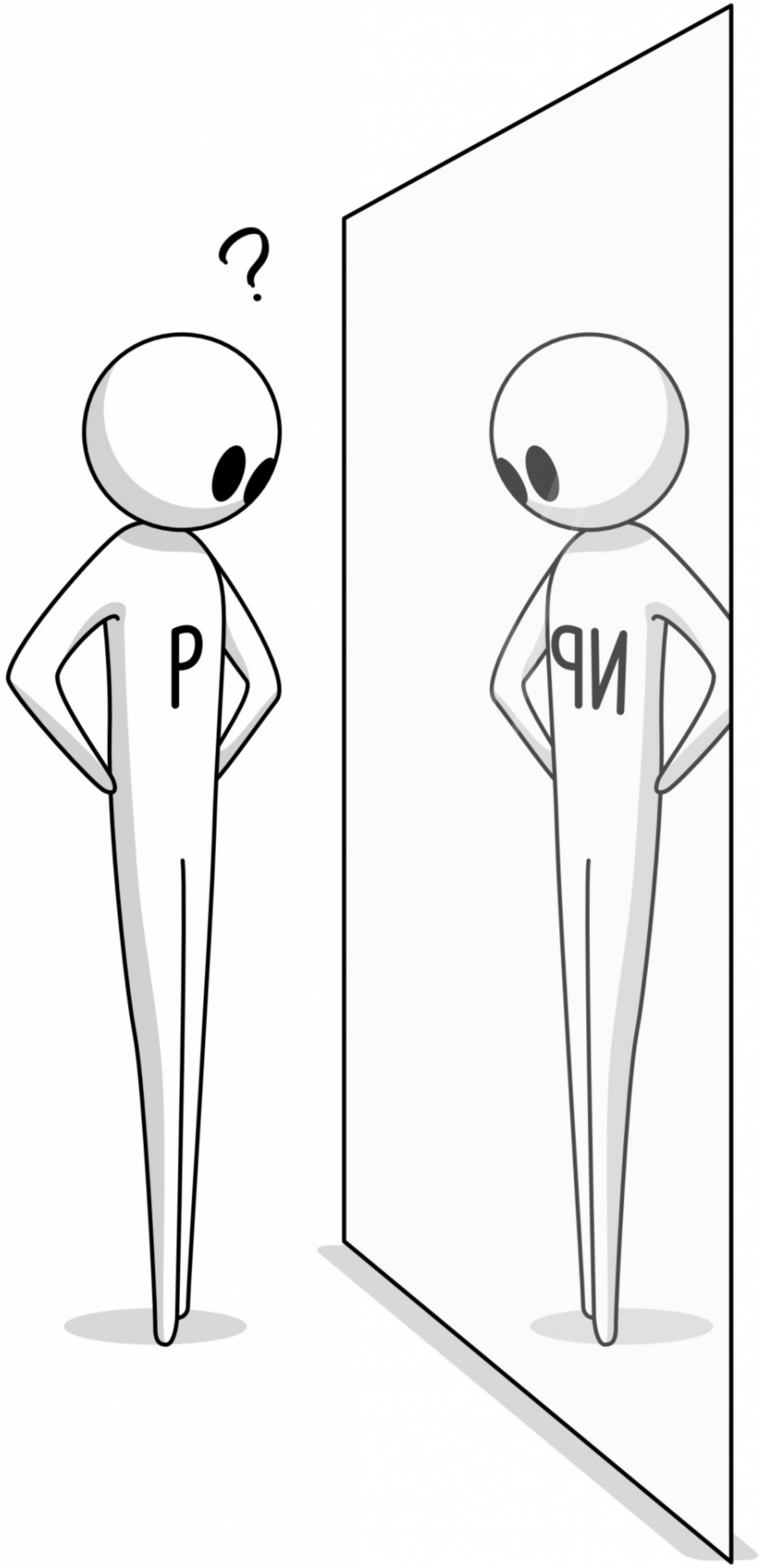
Quelques rappels

- **complexité temporelle d'un algorithme** = nombre d'opérations élémentaires effectuées par celui-ci, dans le pire des cas

- **problème** = ensemble de questions, solubles (ou non) par un algorithme (ensemble infini → problème intéressant!)

Aujourd'hui, nous allons nous intéresser aux **classes de complexité des problèmes**.



Information, Calcul et Communication

Classes de complexité des problèmes

Olivier Lévêque

École polytechnique fédérale de Lausanne

Classes de complexité des problèmes

- **Première question** : Tout problème est-il soluble par un algorithme?

  Réponse : non !

- **Deuxième question** : Les problèmes solubles le sont-ils tous en un temps raisonnable ?

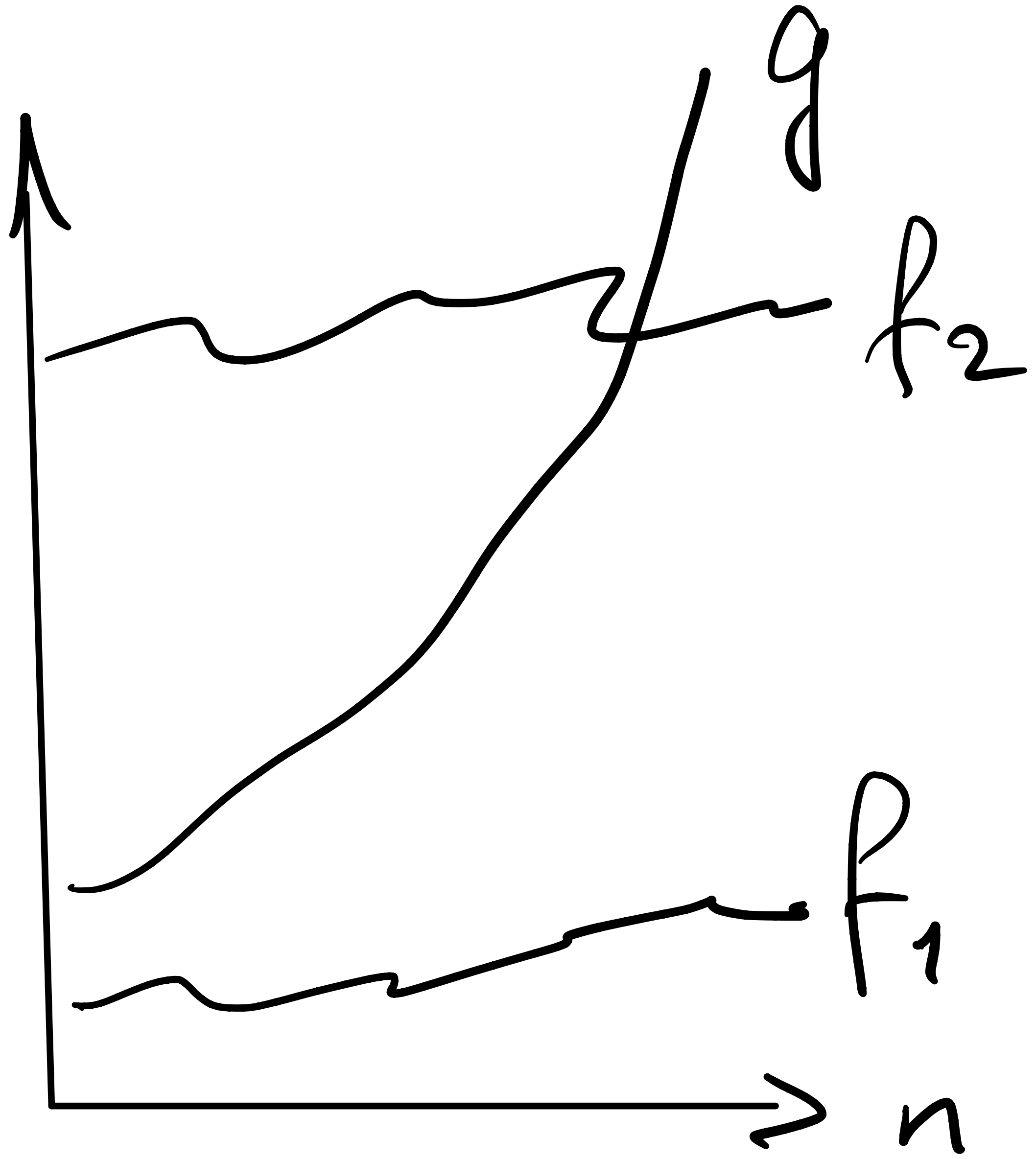
  Réponse : Encore non !

Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

$$f(n) \leq C\, g(n) \text{ pour tout } n \geq N$$



Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

$$f(n) \leq C\, g(n) \text{ pour tout } n \geq N$$

Exemples

- Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
- Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$

$f = \Theta(g)$

ssi $f = O(g)$
& $g = O(f)$

Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

$$f(n) \leq C\, g(n) \text{ pour tout } n \geq N$$

Exemples

- Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
- Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$
- La fonction $f(n) = 3\,n + 1$ est un $O(n)$, est aussi un $O(n^2)$
- La fonction $f(n) = n^2 + 2n + 1$ est un $O(n^2)$, mais n'est pas un $O(n)$

Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

$$f(n) \leq C\, g(n) \text{ pour tout } n \geq N$$

Exemples

- Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
- Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$

- La fonction $f(n) = 3\,n + 1$ est un $O(n)$, est aussi un $O(n^2)$
- La fonction $f(n) = n^2 + 2n + 1$ est un $O(n^2)$, mais n'est pas un $O(n)$

- La fonction $\log_2(n)$ est un $O(n^p)$ pour tout $p > 0$
- La fonction n^p est un $O(2^n)$ pour tout $p > 0$

Deux classes de complexité importantes

Définitions

- La **classe P** est l'ensemble des **problèmes** qui peuvent être **résolus en temps polynomial**, i.e., *pour lesquels il existe un algorithme de résolution dont la complexité temporelle est un $O(n^p)$ pour des données d'entrée de taille n (avec $p \geq 1$ un nombre fixé).*

- La **classe NP** est l'ensemble des **problèmes** pour lesquels, *si "on" nous propose une solution du problème, il est alors possible de* ***vérifier en temps polynomial*** *si celle-ci en est une ou pas.*

Remarques

- NP ne veut **pas** dire "non-polynomial"
- P $\subset$ NP (il est plus facile de vérifier une solution que d'en proposer une !)

Exemples de problèmes appartenant à la classe P

Beaucoup de problèmes que nous avons déjà rencontrés dans ce cours appartiennent à la classe P (et sont donc des problèmes "faciles" à résoudre) :

- Le problème de la recherche d'un élément dans une liste de taille n.
- Le problème du tri d'une liste de taille n.
- Le calcul de la somme ou de la moyenne de n nombres.
- Identifier si tous les éléments d'une liste (de taille n) sont différents.

Tous ces problèmes admettent en effet des algorithmes de résolution de complexité temporelle $\Theta(n)$, $\Theta(n \cdot \log_2(n))$ ou $\Theta(n^2)$,

Exemples de problèmes appartenant à la classe P

Beaucoup de problèmes que nous avons déjà rencontrés dans ce cours appartiennent à la classe P (et sont donc des problèmes "faciles" à résoudre) :

- Le problème de la recherche d'un élément dans une liste de taille n.
- Le problème du tri d'une liste de taille n.
- Le calcul de la somme ou de la moyenne de n nombres.
- Identifier si tous les éléments d'une liste (de taille n) sont différents.

Tous ces problèmes admettent en effet des algorithmes de résolution de complexité temporelle $\Theta(n)$, $\Theta(n \cdot \log_2(n))$ ou $\Theta(n^2)$, *donc* $O(n^2)$.

Exemples de problèmes appartenant à la classe NP

- Tous les problèmes mentionnés avant, du fait qu'ils appartiennent à la classe P, appartiennent également à la classe NP !

- Voici maintenant un autre problème appartenant à la classe NP:

  « Soient P, Q deux nombres premiers à n chiffres, et soit $N = P \cdot Q$.
  Etant donné N, on aimerait retrouver P et Q. »

  Exemple: si $N = 98'201$, que valent P et Q ?

- Ce problème n'est a priori pas facile à résoudre… Par contre, si on nous propose une solution (ici, $P = 347$ et $Q = 283$), il est alors facile de vérifier que $N = P \cdot Q$ en effectuant la multiplication (en $\Theta(n^2)$ opérations) : le problème appartient donc à la classe NP.

EPFL

Exemples de problèmes

1. « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il **deux indices** différents $i, j \in \{1, \dots, n\}$ tels que $L(i) + L(j) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23) \rightarrow$ Réponse : non

   Comme nous l'avons déjà vu, un algorithme de complexité temporelle $\Theta(n^2)$, ou même $\Theta(n)$, permet de répondre à cette question: le problème ci-dessus fait donc partie de la **classe P**.

Exemples de problèmes

2. « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il **trois indices** différents $i, j, k \in \{1, \ldots, n\}$ tels que $L(i) + L(j) + L(k) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23) \rightarrow$ Réponse : encore non!

   Un algorithme de complexité temporelle $\Theta(n^3)$ permet de répondre à cette question: le problème ci-dessus fait donc également partie de la **classe P**.

$\Theta(n^2)$

Exemples de problèmes

3. « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il **un sous-ensemble** $S \subset \{1, \ldots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$

   Réponse : oui ! En effet : $-15 \ - 12 - 1 + 5 + 23 = 0$

$S = \{1, 2, 4, 5, 7\}$

$E = \{1, 2, \ldots, n\} \longrightarrow$ nb de sous-ensembles de E

$= \underline{\underline{2^n}} = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{\underline{\underline{n \text{ fois}}}}$

Exemples de problèmes

3. « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il **un sous-ensemble** $S \subset \{1, \dots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$

   Réponse : oui ! En effet : $-15\ -12 - 1 + 5 + 23 = 0$

   Mais pour obtenir cette réponse, on ne connaît pas d'autre algorithme que de tester tous les sous-ensembles S possibles, qui sont au nombre de 2^n.
   Conclusion 1: On ne sait pas si ce problème fait partie de la **classe P**.

   Par contre, si on nous propose une solution du problème, il est alors facile de **vérifier** en temps polynomial que c'en est bien une! (additionner n nombres ne nécessite que $\Theta(n)$ opérations).
   Conclusion 2: Le problème ci-dessus fait partie de la **classe NP**.

Et pour finir, une question à un million de dollars !

- Le problème :

  « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il un sous-ensemble $S \subset \{1, \ldots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

  s'appelle le **problème des sommes de sous-ensembles**. On peut montrer qu'il fait partie des problèmes **les plus difficiles** à résoudre parmi ceux de la classe NP.

  = Problème NP - complet

Et pour finir, une question à un million de dollars !

- Le problème :

  « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

  s'appelle le **problème des sommes de sous-ensembles**. On peut montrer qu'il fait partie des problèmes **les plus difficiles** à résoudre parmi ceux de la classe NP.

- Si on arrive à démontrer un jour que ce problème fait **aussi** partie de la classe P, ou au contraire qu'il n'en fait **pas** partie, on aura alors résolu la question de savoir si **P=NP** ou non, pour laquelle le Clay Mathematics Institute a promis une récompense d'une valeur d'un million de dollars.

EPFL

Information, Calcul et Communication

Le problème du sac à dos

Olivier Lévêque

École polytechnique fédérale de Lausanne

Problèmes d'optimisation discrète

Exemple : former un comité

$$E = \{1, \dots, n\}$$

$\longrightarrow$ $\cdot\ S = \{i\}$

$\cdot\ S = E$

$\cdot\ S = \{i_1, \dots, i_m\}$

Problèmes d'optimisation discrète

- **Généralisation :**

  « Etant donnée une **fonction** f qui à tout sous-ensemble donné $S \subset \{1, \dots, n\}$ associe une valeur réelle $f(S)$, trouver un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $f(S)$ soit minimale. »

- Pour de nombreuses fonctions f, et donc pour de nombreux **problèmes d'optimisation discrète**, on ne sait *pas* s'ils appartiennent à la classe P, *ni* s'ils appartiennent à la classe NP.

- Tous ces problèmes sont caractérisés par l'existence d'un **nombre fini, mais grand,** de solutions possibles.

Classes de complexité des problèmes : résumé

(ex : sommes de ss-ens.)

NP-complets

(ex : tri)

P

problèmes indécidables

difficulté

NP

NP-difficiles

(ex : factorisation)

(ex : pbs d'optimisation discrète)

Le problème du sac à dos

- n objets, chacun pesant un certain poids
- un sac à dos de capacité maximale C
- Comment remplir au mieux le sac à dos?

Le problème du sac à dos

« Etant donnés une liste L de n nombres entiers positifs et $C > 0$, trouver un sous-ensemble $S \subset \{1, \dots, n\}$ tel que

$$\sum_{i \in S} L(i) \leq C$$

et

$\sum_{i \in S} L(i)$ soit maximale.»

Le problème du sac à dos

« Etant donnés une liste L de n nombres entiers positifs et $C > 0$, trouver un sous-ensemble $S \subset \{1, \ldots, n\}$ tel que

$$\sum_{i \in S} L(i) \leq C$$

et

$$\sum_{i \in S} L(i) \text{ soit maximale.»}$$

Hypothèses additionnelles:

- $L(i) \leq C$ pour toute valeur de i
- $\sum_{i=1}^{n} L(i) > C$

Le problème du sac à dos

= problème d'optimisation discrète (avec contrainte)

On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP.

Le problème du sac à dos

= problème d'optimisation discrète (avec contrainte)

On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP.

Voici toutefois un algorithme polynomial en n qui remplit au moins la moitié du sac à dos :

Le problème du sac à dos

= problème d'optimisation discrète (avec contrainte)

On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP.

Voici toutefois un algorithme polynomial en n qui remplit au moins la moitié du sac à dos :

1. Ordonner la liste L dans l'ordre décroissant

Le problème du sac à dos

= problème d'optimisation discrète (avec contrainte)

On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP.

Voici toutefois un algorithme polynomial en n qui remplit au moins la moitié du sac à dos :

1. Ordonner la liste L dans l'ordre décroissant
2. Chercher le nombre $k \in \{1, \ldots, n-1\}$ tel que

$$\sum_{i=1}^{k} L(i) \leq C \qquad \text{et} \qquad \sum_{i=1}^{k+1} L(i) > C$$

Le problème du sac à dos

= problème d'optimisation discrète (avec contrainte)

On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP.

Voici toutefois un algorithme polynomial en n qui remplit au moins la moitié du sac à dos :

1. Ordonner la liste L dans l'ordre décroissant
2. Chercher le nombre $k \in \{1, \dots, n-1\}$ tel que

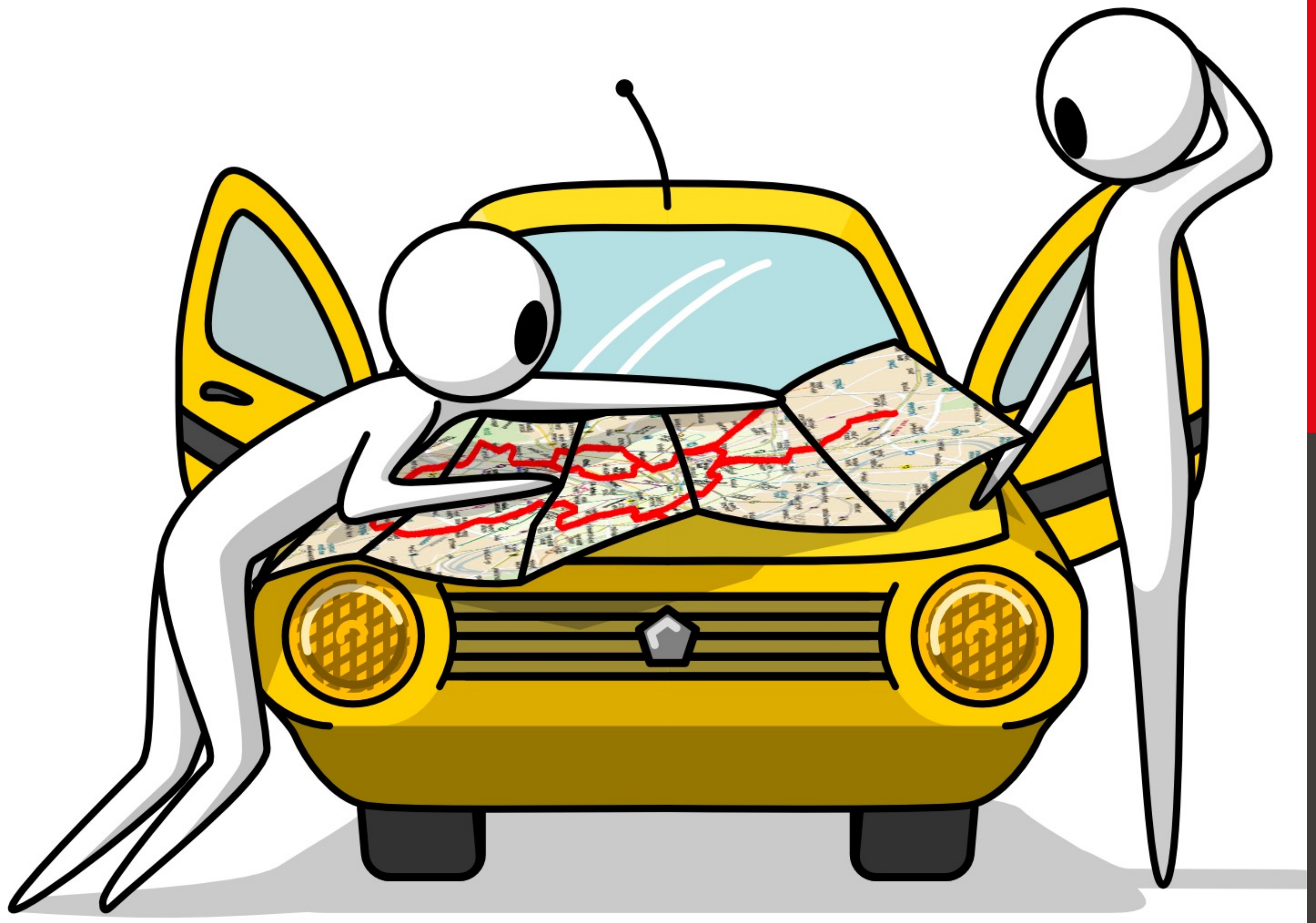
$$\sum_{i=1}^{k} L(i) \leq C \qquad \text{et} \qquad \sum_{i=1}^{k+1} L(i) > C$$

Alors $\sum_{i=1}^{k} L(i) \geq C/2$;

supposons le contraire, à savoir : $\boxed{\sum_{i=1}^{k} L(i) < \frac{C}{2}}$

alors $\sum_{i=1}^{k+1} L(i) = \underbrace{\sum_{i=1}^{k} L(i)} + \underbrace{L(k+1)}_{\leq L(k) \leq \sum_{i=1}^{k} L(i)} < \frac{C}{2} + \frac{C}{2} = C$ ↯

EPFL



Information, Calcul et Communication

Le problème du voyageur de commerce

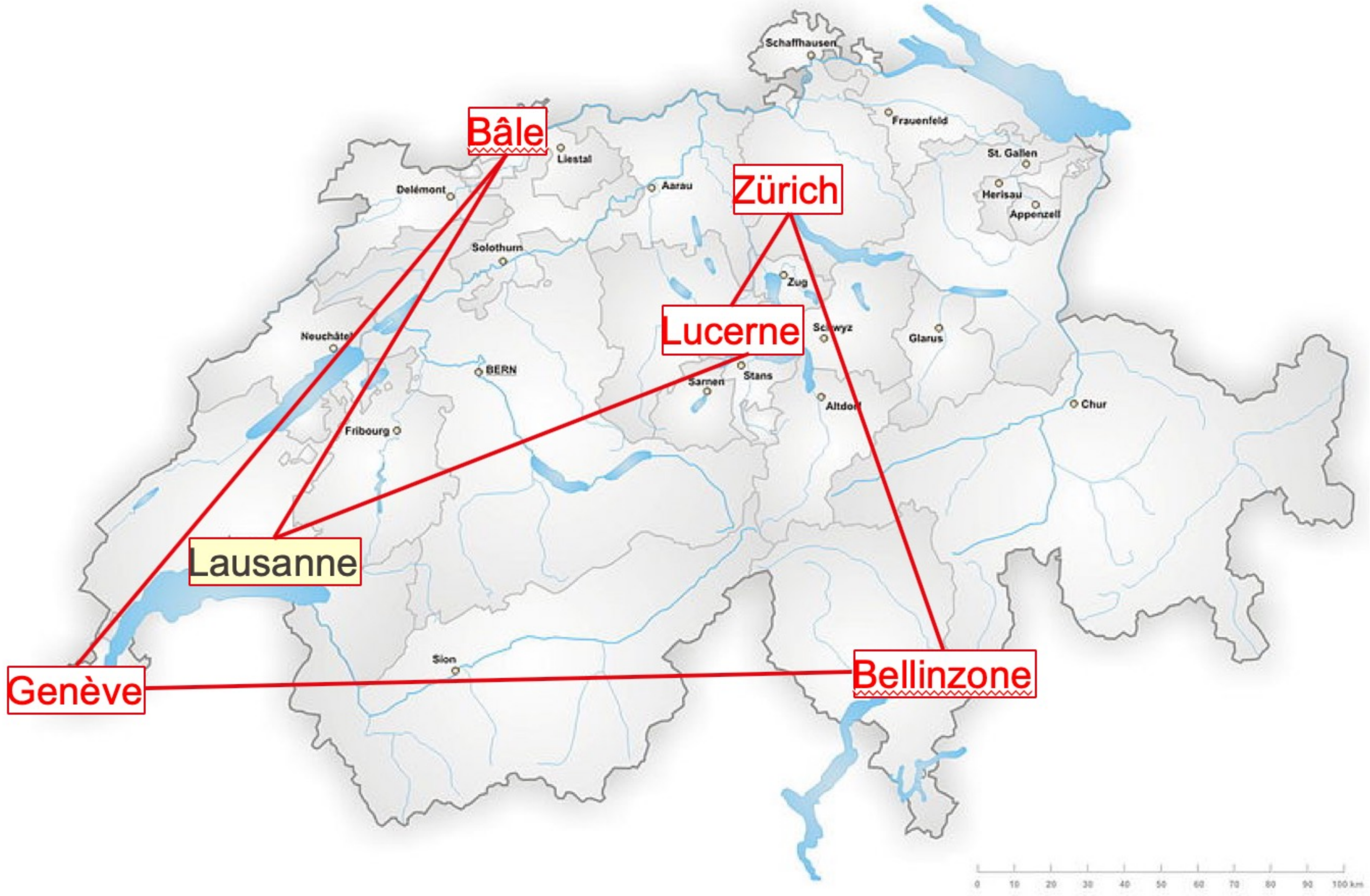
Olivier Lévêque

École polytechnique fédérale de Lausanne

Le problème du voyageur de commerce

Version « euclidienne »:

Etant donné un ensemble de n villes sur une carte, trouver le chemin fermé le plus court passant une et une seule fois par chacune de ces villes.



$n!$ possibilités

$\sim n^n$

Note : On suppose ici des voyages à vol d'oiseau entre chaque ville.

Algorithmes de résolution

Premier essai :

Tester tous les chemins fermés possibles

➔ Cet algorithme trouve le chemin fermé optimal ✓

… mais les chemins fermés sont au nombre de $n! = n(n-1)(n-2)\dots1$

impossible à mettre en pratique !

Algorithmes de résolution

Deuxième essai :

1. Partir d'une ville choisie au hasard
2. Tant qu'il reste une ville non-explorée, rejoindre la ville non-explorée la plus proche
3. Quand toutes les villes ont été explorées, revenir à la ville de départ

➔ complexité temporelle polynomiale en n ✓

… mais le chemin fermé résultant est loin d'être optimal en général !

Algorithmes de résolution

Troisième essai :

1. Choisir un **chemin fermé** $(v_1, v_2, \ldots, v_n, v_{n+1} = v_1)$ au hasard
2. Effectuer une boucle pour i allant de 1 à n :

   Permuter les villes v_i et v_{i+1} dans le chemin si cette permutation raccourcit la longueur totale du chemin

➔ complexité temporelle polynomiale en n ✓

… meilleure performance moyenne que l'algorithme précédent, mais résultat toujours aléatoire

Algorithme de résolution avec garantie d'approximation (et complexité temporelle polynomiale en n)

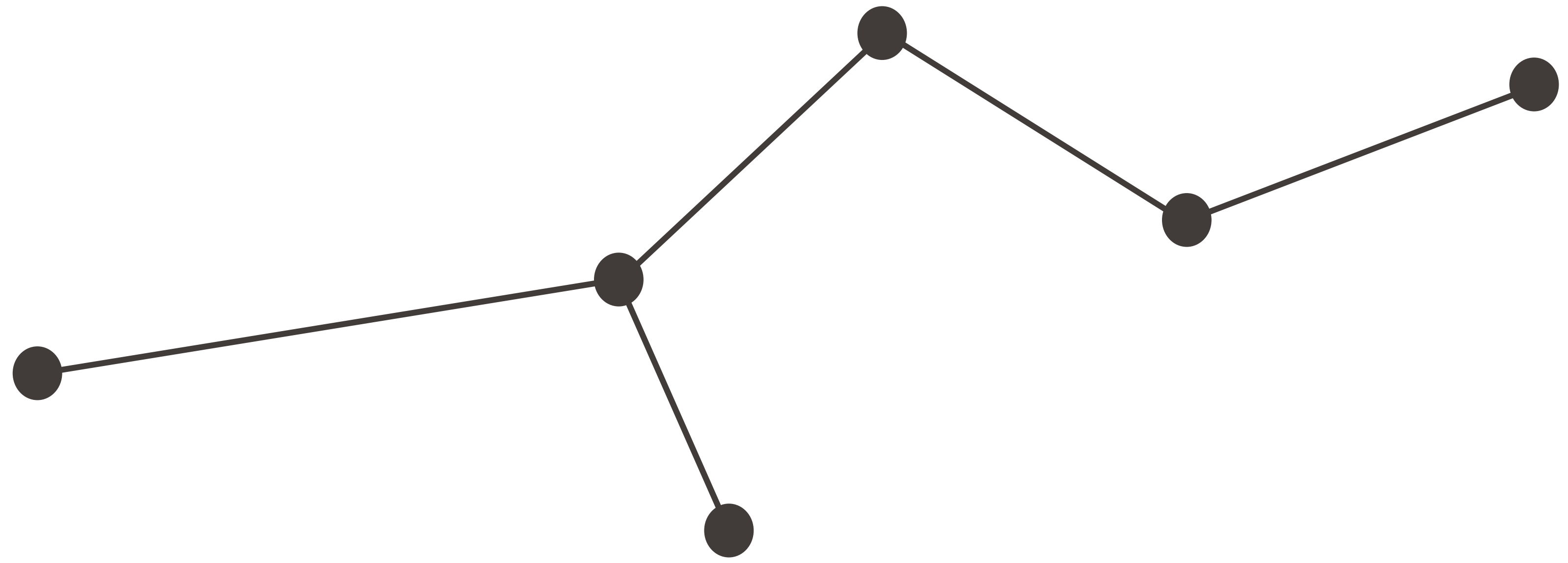
Théorème

Soit L_{min} la longueur du chemin fermé optimal (i.e., le plus court).
Il existe un algorithme de complexité temporelle polynomiale en n permettant de trouver un chemin fermé de longueur $L \leq 2L_{min}$.

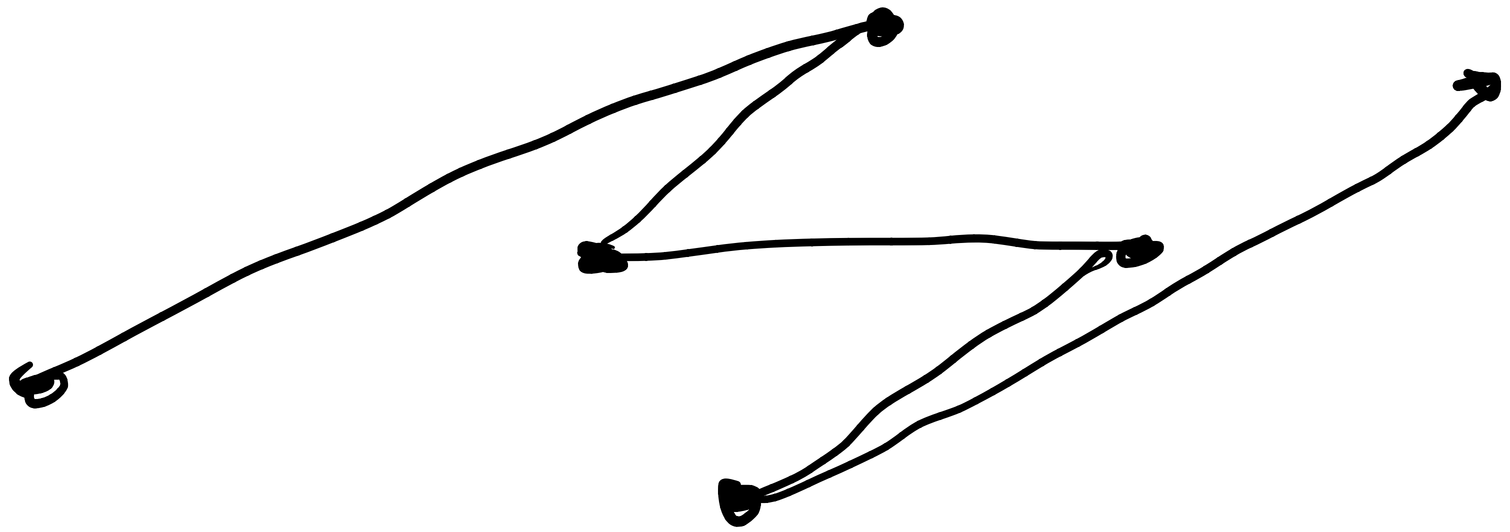
Première étape : arbre couvrant minimal

Etant données des villes sur une carte, on cherche à les relier par un arbre dont la somme des longueurs des branches soit minimale.

Exemple :



arbre couvrant minimal

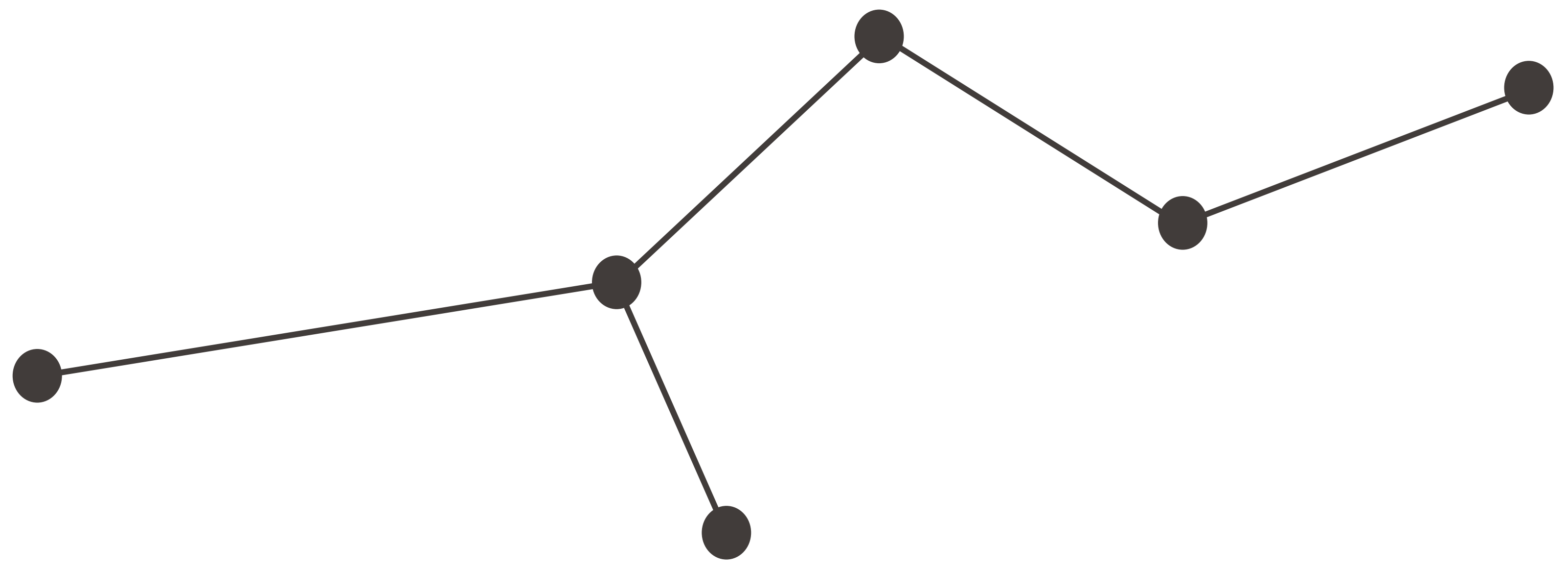


arbre couvrant

Première étape : arbre couvrant minimal

Etant données des villes sur une carte, on cherche à les relier par un arbre dont la somme des longueurs des branches soit minimale.

Exemple :

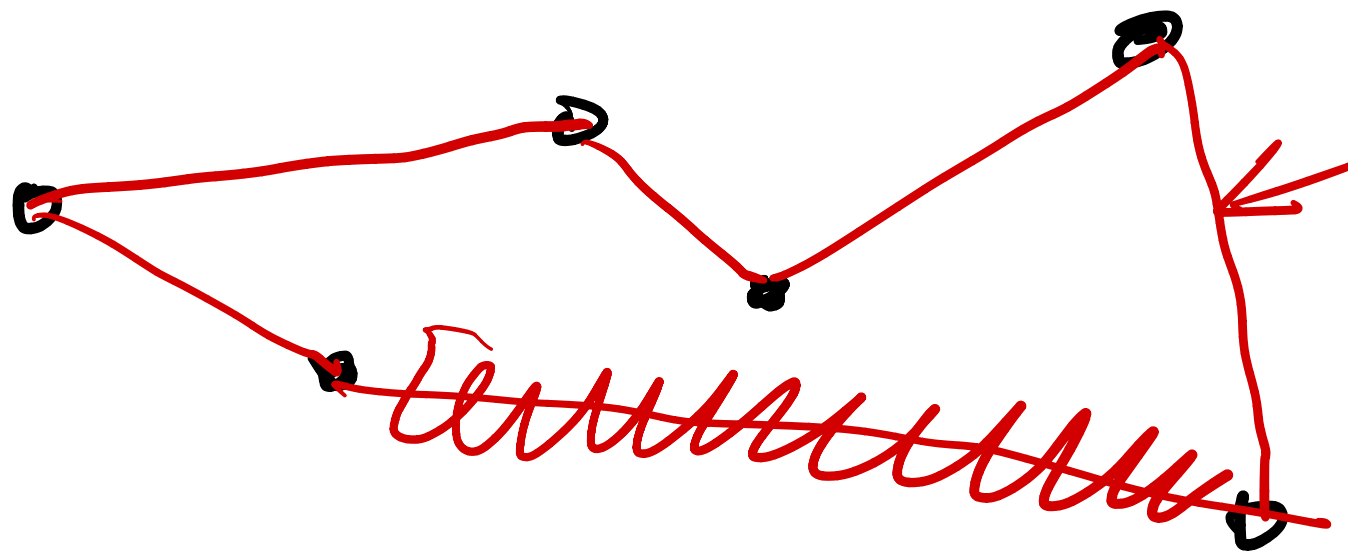


Il est possible de trouver un tel arbre (T) en temps polynomial en n :

1. Commencer avec $T = \{$une ville au hasard$\}$ (= racine de l'arbre)
2. Chercher parmi les villes restantes la ville v qui soit la plus proche d'une des villes $w \in T$: rajouter v et la branche $v - w$ à T
3. Recommencer en 2. jusqu'à ce que T contienne toutes les villes

Remarque importante

Si L_T désigne la somme des longueurs des branches de l'arbre couvrant minimal, et L_{min} désigne la longueur du chemin fermé optimal qui passe une fois par chaque ville, alors $\boxed{L_{min} \geq L_T\ .}$

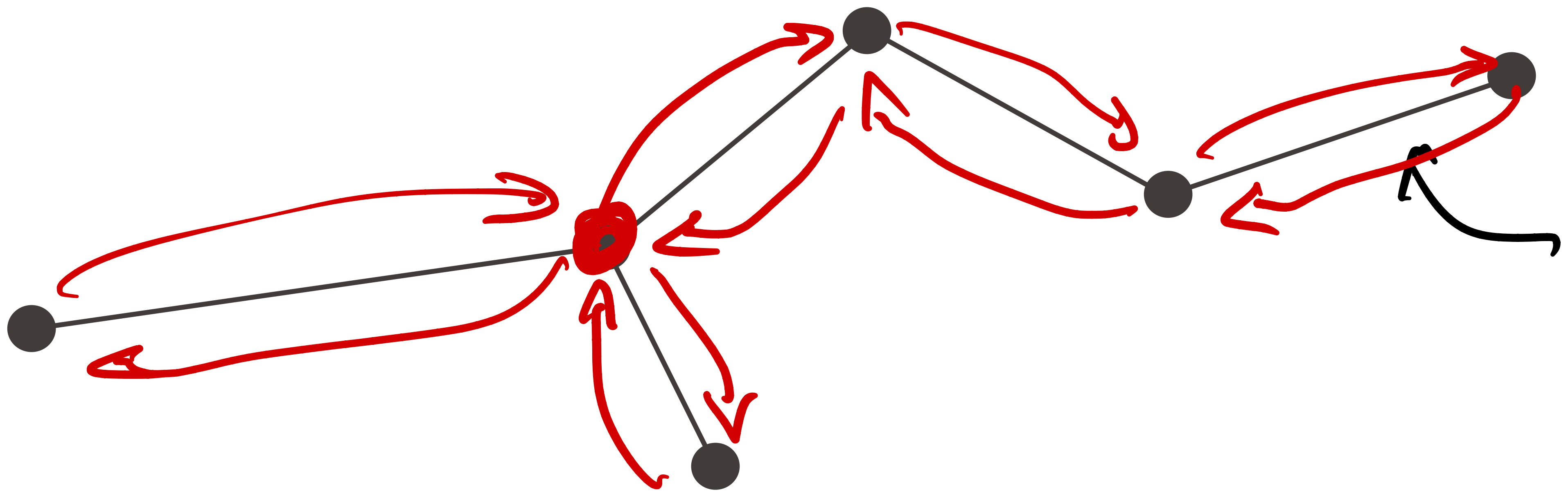


chemin fermé optimal

= arbre couvrant

Deuxième étape : parcours le long de l'arbre

Illustration

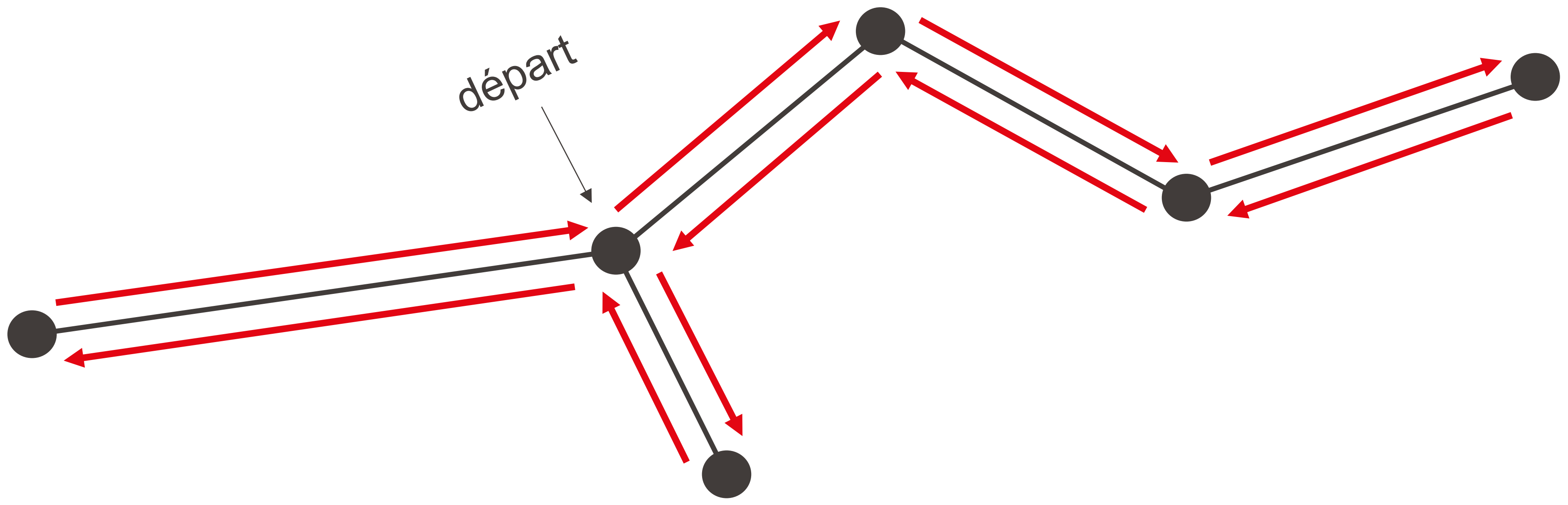


arbre couvrant minimal

$$L(\text{chemin rouge}) = 2 \cdot L_T \leq 2 \cdot L_{min}$$

Deuxième étape : parcours le long de l'arbre

Illustration



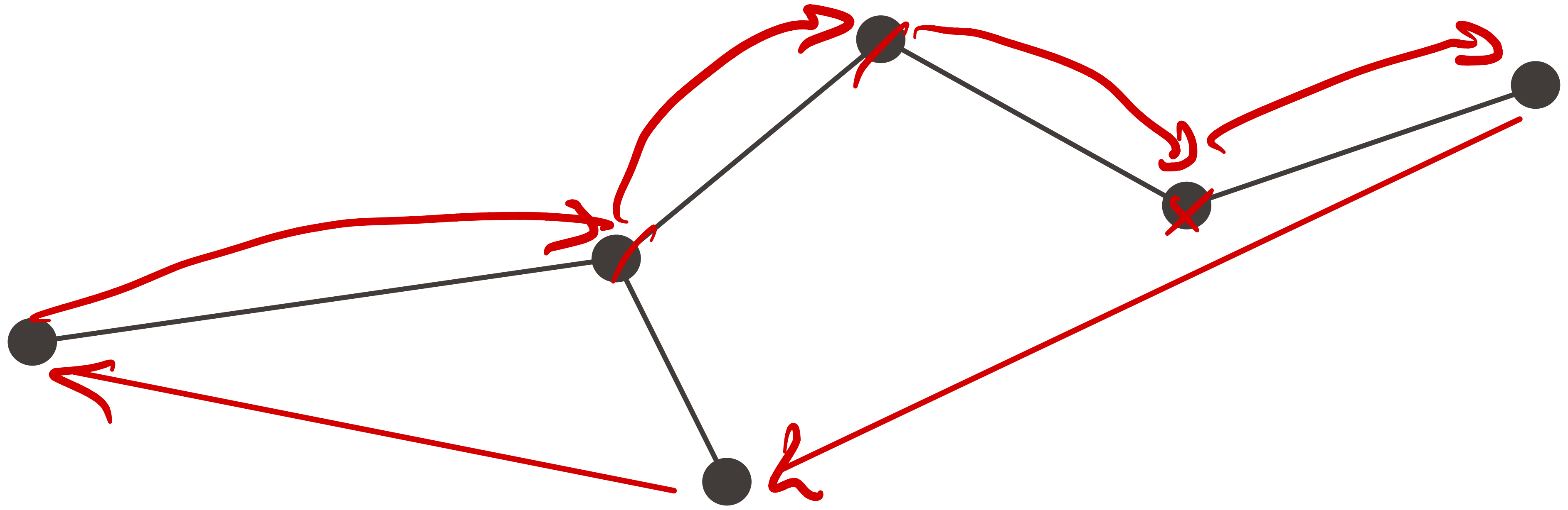
La longueur du chemin rouge vaut $2L_T \leq 2L_{min}$

Presque ce qu'on veut, mais le chemin rouge n'est pas un chemin qui passe une et une seule fois par chaque ville.

Troisième et dernière étape : prendre des raccourcis

On modifie le chemin rouge en suivant la règle : dès qu'une ville à déjà été visitée, on prend un raccourci vers la ville suivante dans l'arbre.

Illustration

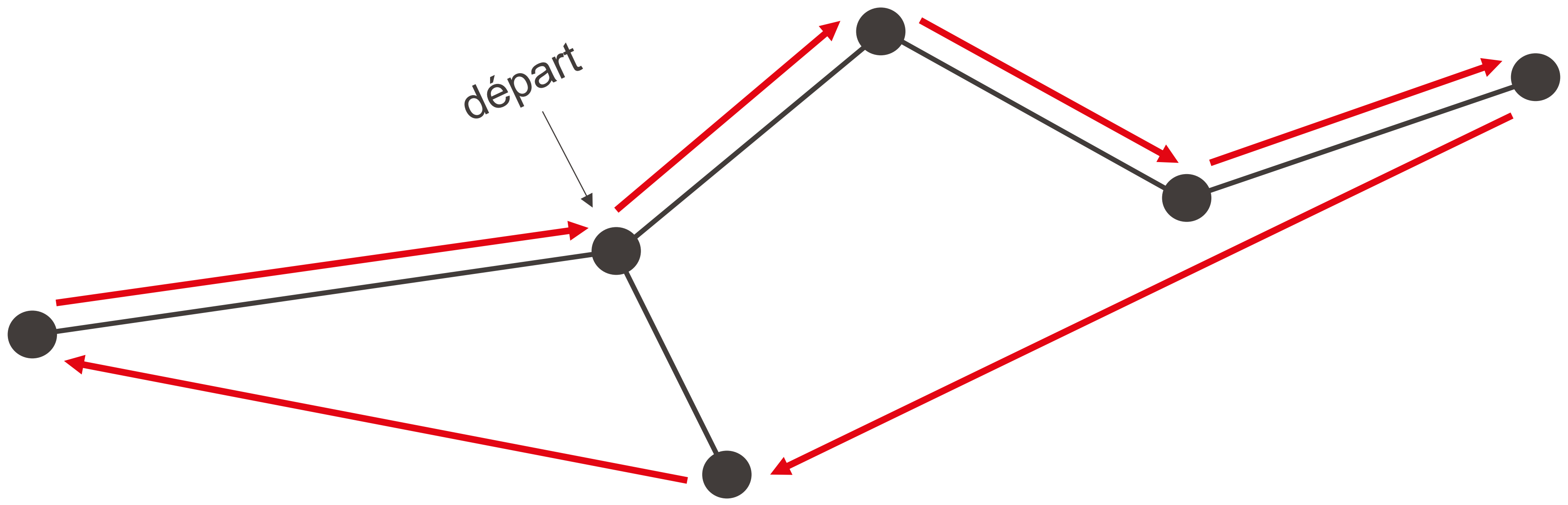


Donc $L(\text{ce chemin rouge}) \leq 2 \cdot L_{min}$

Troisième et dernière étape : prendre des raccourcis

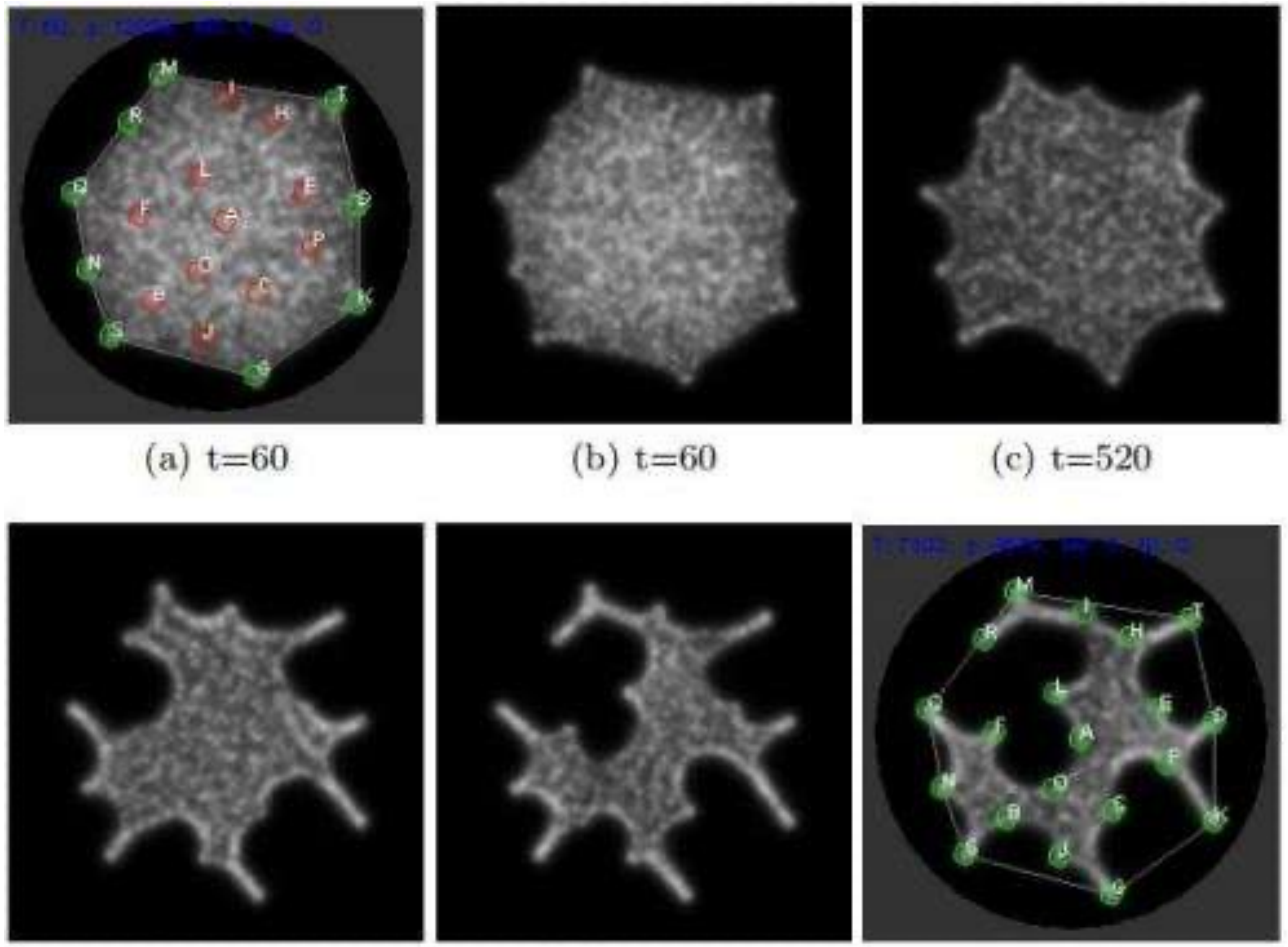
On modifie le chemin rouge en suivant la règle : dès qu'une ville à déjà été visitée, on prend un raccourci vers la ville suivante dans l'arbre.

Illustration



Ce nouveau chemin rouge n'est pas plus long que le précédent (donc $\leq 2L_{min}$), mais il remplit cette fois la condition de passer une fois dans chaque ville. $\#$

Et pour finir, en voilà un qui résout le problème tout seul!



(a) t=60 (b) t=60 (c) t=520

(d) t=3360 (e) t=6360 (f) t=7102

Source: “Computation of the Travelling Salesman Problem by a Shrinking Blob” https://arxiv.org/abs/1303.4969

Il s’agit d’un “blob” (plus savamment, un physarum polycephalum), organisme unicellulaire capable d’optimiser sa structure interne pour accéder à de la nourriture.