

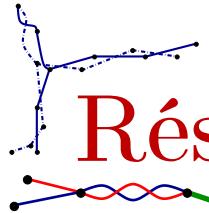
Conception et Pratique de l'Algorithmique

<http://www-apr.lip6.fr/~buixuan/cpa2020>

Binh-Minh Bui-Xuan



PARIS, Avril 2021



Résumé

Take home message:

Informatic → arbre
(TCS, ST)



$\mathcal{O}(n)$

CONSTRUCTION D'ARBRE :

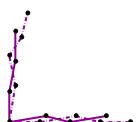
NP-complet

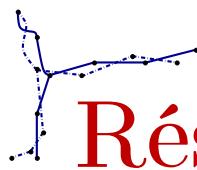
- arbre couvrant, arbre de Steiner, décomposition arborescente
- programmation dynamique, recherche locale
- concours de programmation

DP - LS

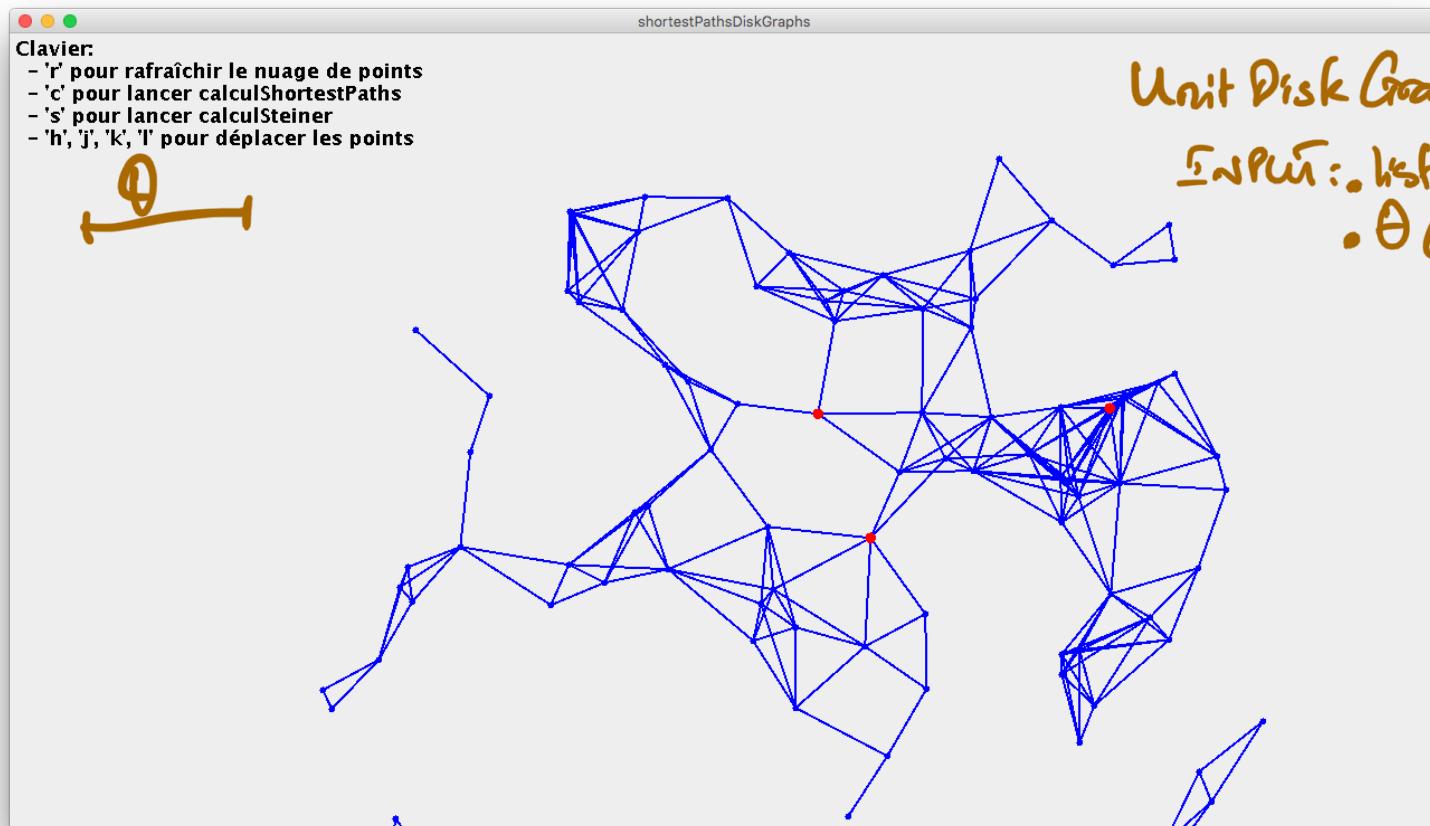


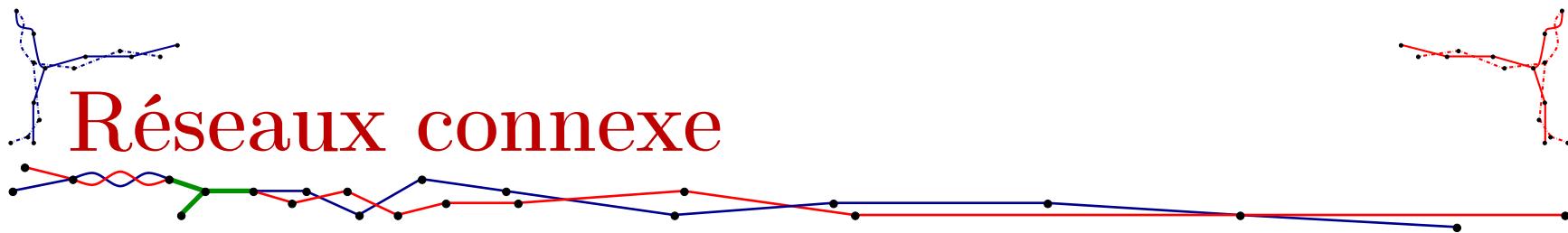
→ probablement
en $\mathcal{O}(n^2)$



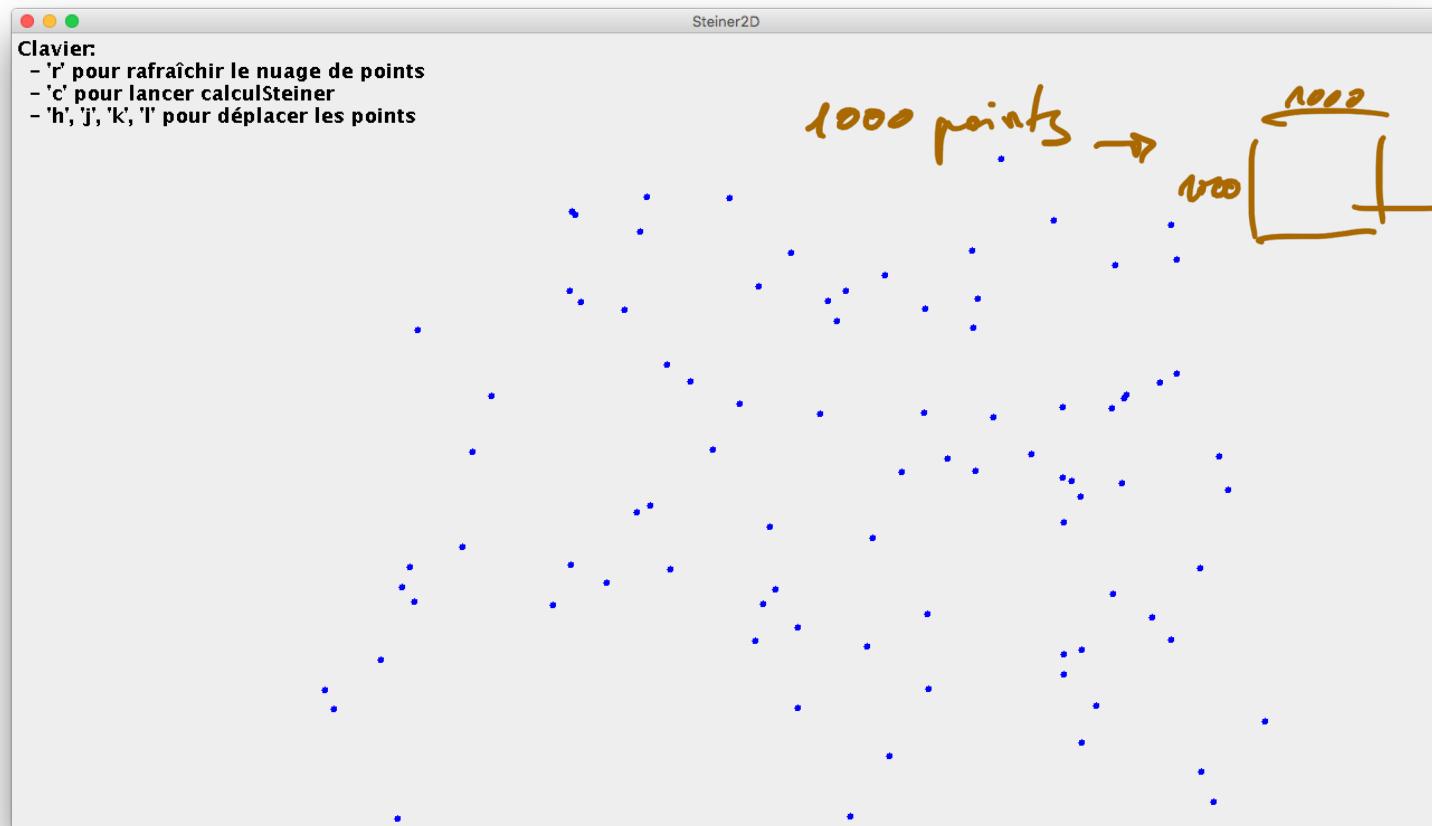


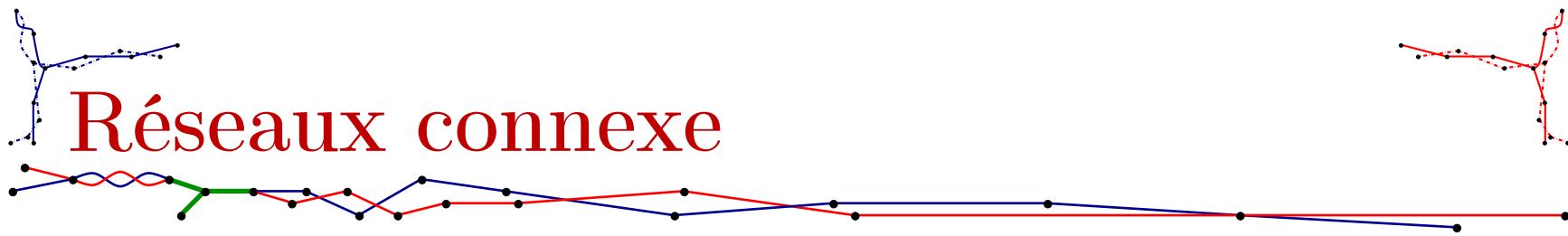
Réseaux connexes



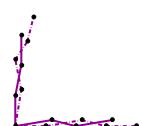
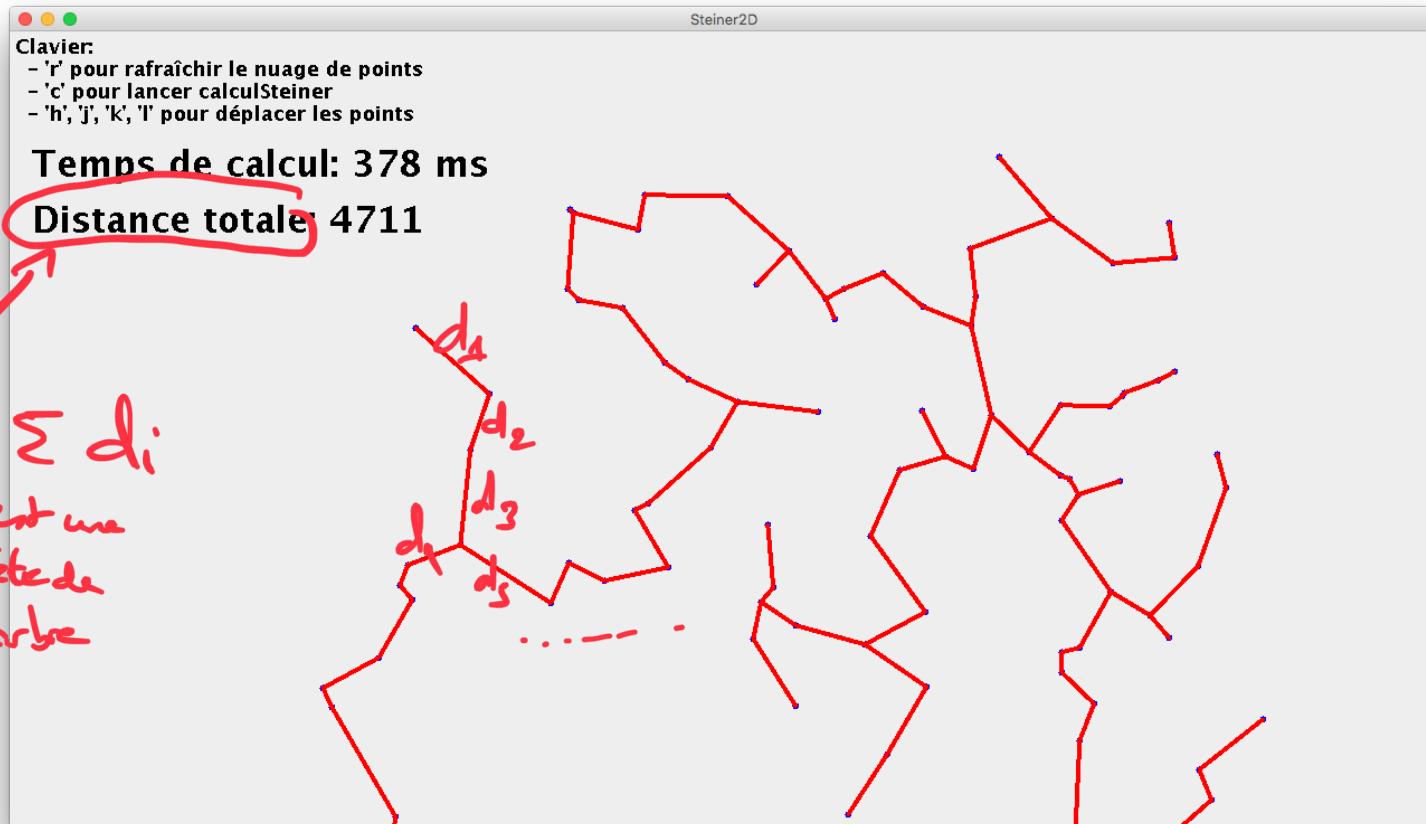


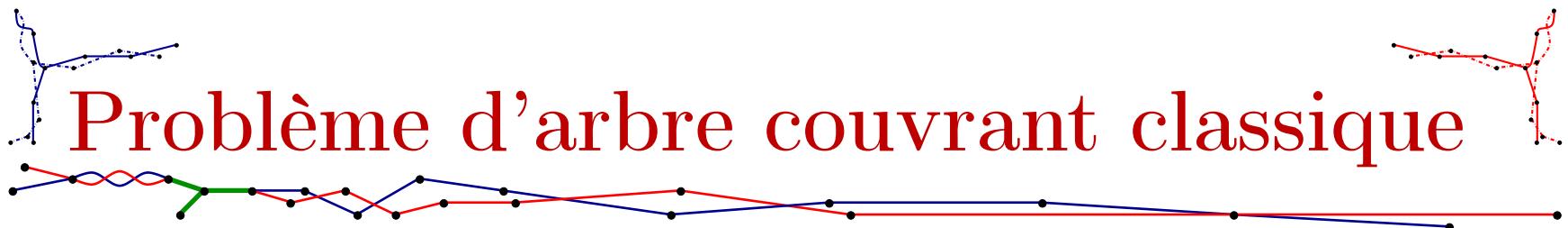
Réseaux connexes





Réseaux connexes



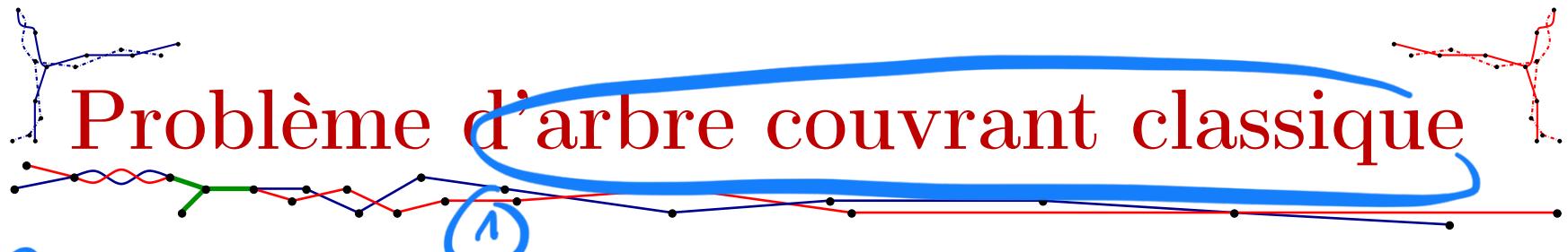


Problème d'arbre couvrant classique

IN : Points, une liste de coordonnées de points en 2D

OUT : arbre MSTree couvrant tous les points de la liste, de poids total minimum (somme de la distance entre toutes les arêtes). De plus, tous les sommets de MSTree doivent appartenir à la liste Points.





IN : Points, une liste de coordonnées de points en 2D

③ OUT : arbre MSTree couvrant tous les points de la liste, de poids total minimum (somme de la distance entre toutes les arêtes). De plus, tous les sommets de MSTree doivent appartenir à la liste Points.

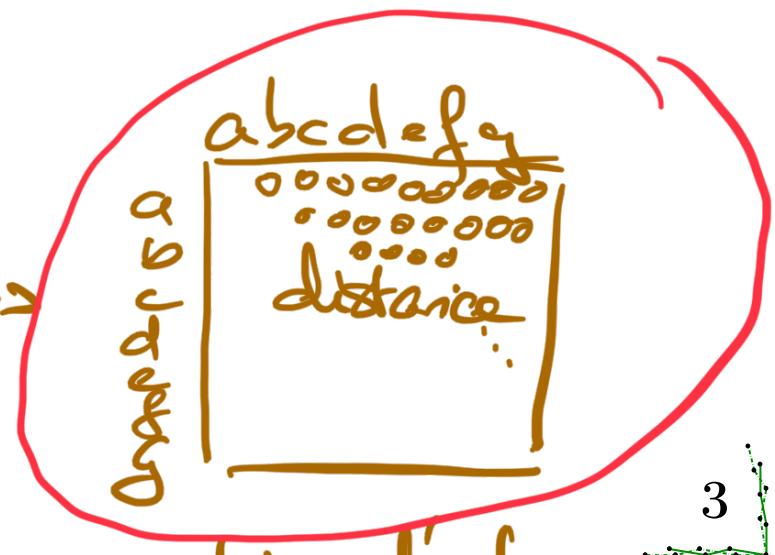
EXERCICE : Structure de graphe ?

Plan Euclidien

\mathbb{R}^2

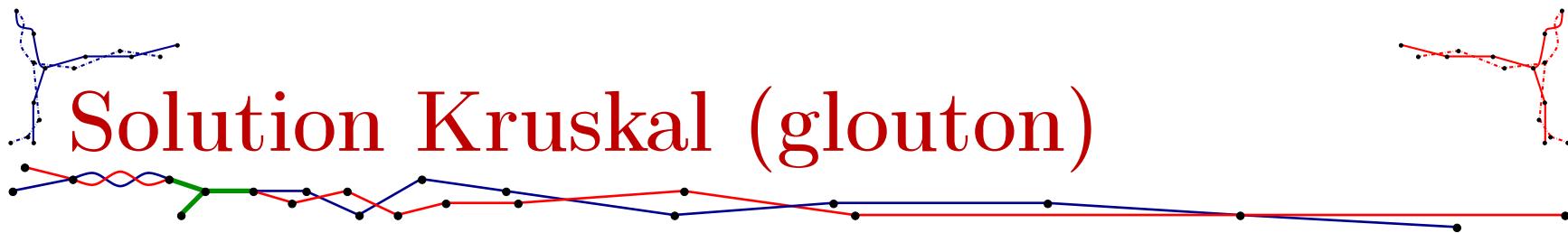


b e
a o e o f
c o d o g



matrice d'adjacence
d'un graphe pondéré.

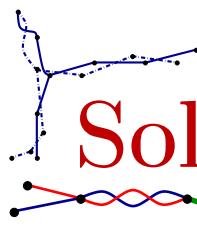
3



PRINCIPE : trier les arêtes par ordre croissant selon leur poids. Tant que l'ajout d'une arête de cette liste ne crée pas de cycle dans la solution (initialement vide), effectuer cet ajout dans la solution.

h
+
reduce





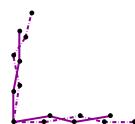
Solution Kruskal (glouton)

Input: ArrayList<Point> points

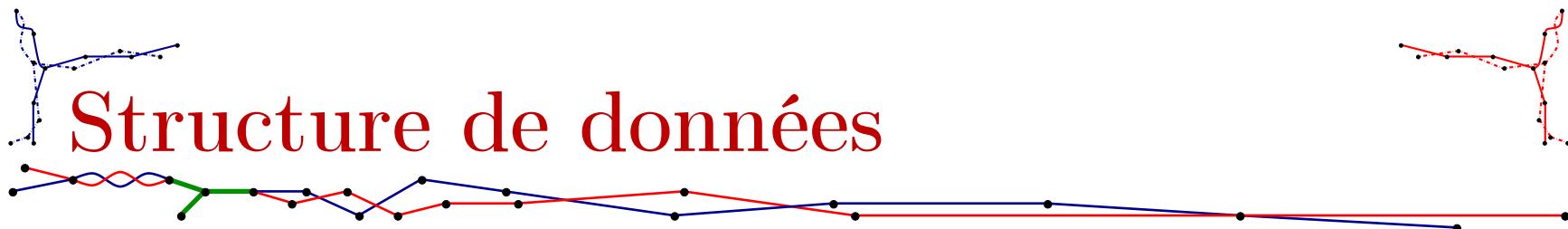
PRINCIPE : trier les arêtes par ordre croissant selon leur poids. Tant que l'ajout d'une arête de cette liste ne crée pas de cycle dans la solution (initialement vide), effectuer cet ajout dans la solution.

EXERCICE : Pseudo-code ?

Rq: liste d'arcs !!!



```
listArcs = calc11(points)
listArcts. tri ();
solution = liste vide
while listArcts.isNotEmpty()
    ajout = solution + listArcts.premier()
    if (ajout.pasDeCycle())
        Solution.add(ajout)
    listArcts.supprime(ajout)
    ?? Condition d'arcs ??
```



Structure de données

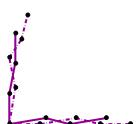
Graphe :

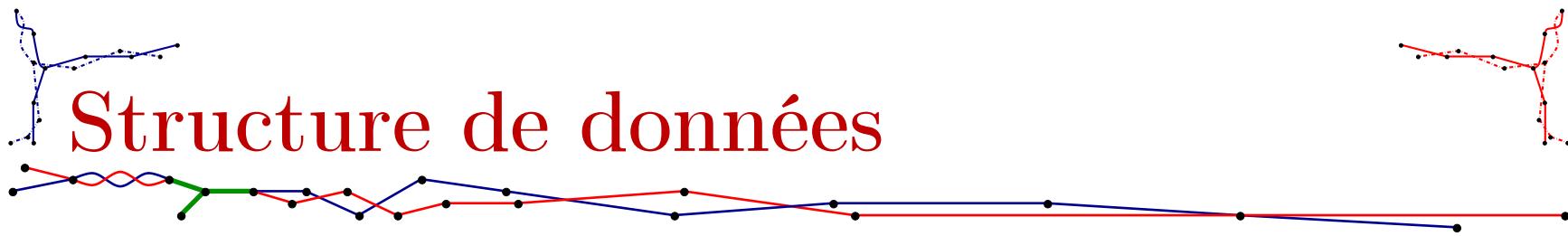
- matrice d'adjacence
- liste d'adjacence
- liste des arêtes

Arbre : (*cas particulier d'un graphe*)

- matrice d'adjacence
- liste d'adjacence
- liste des arêtes
- hiérarchie (noeud, sousarbres)

+
 liste de pointeurs vers d'autres arbres





Structure de données

Graphe :

- matrice d'adjacence
- liste d'adjacence

– liste des arêtes



Arbre :

– matrice d'adjacence

– liste d'adjacence

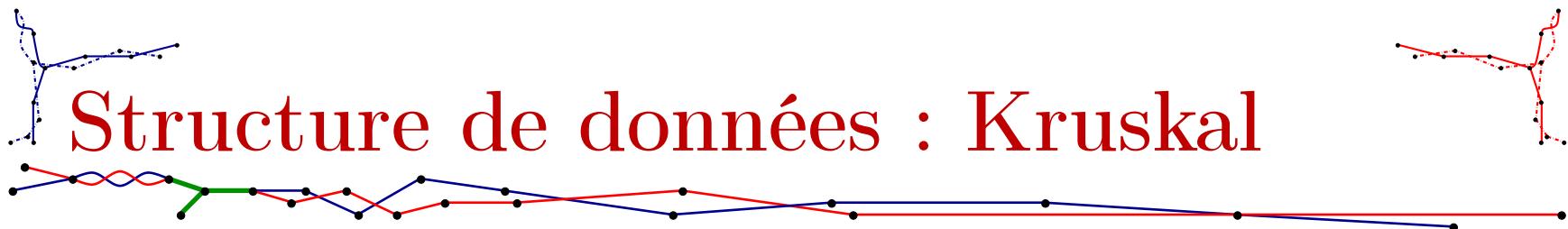
~~– liste des arêtes~~

- hiérarchie (noeud, sousarbres)

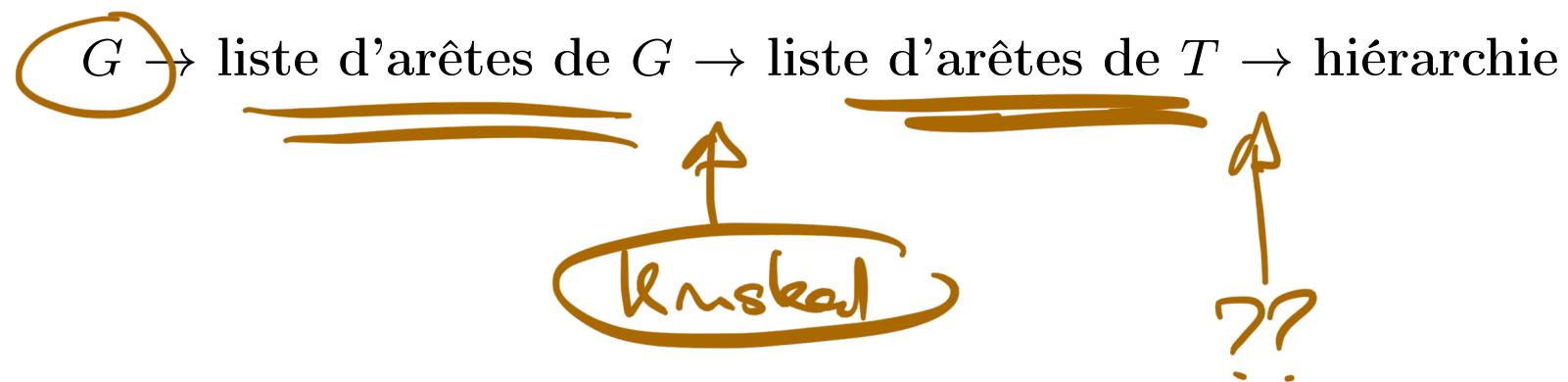
```
public Tree2D calculate(ArrayList<Point> points);
```

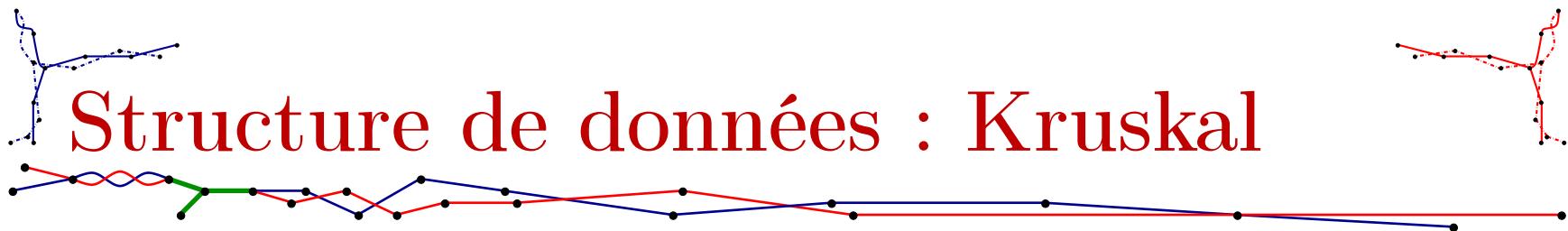
Signature de la
fonction à implémenter
pour Tree2D.





PRINCIPE : avec graphe G en input et arbre T en output :





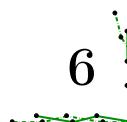
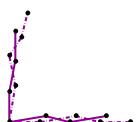
Structure de données : Kruskal

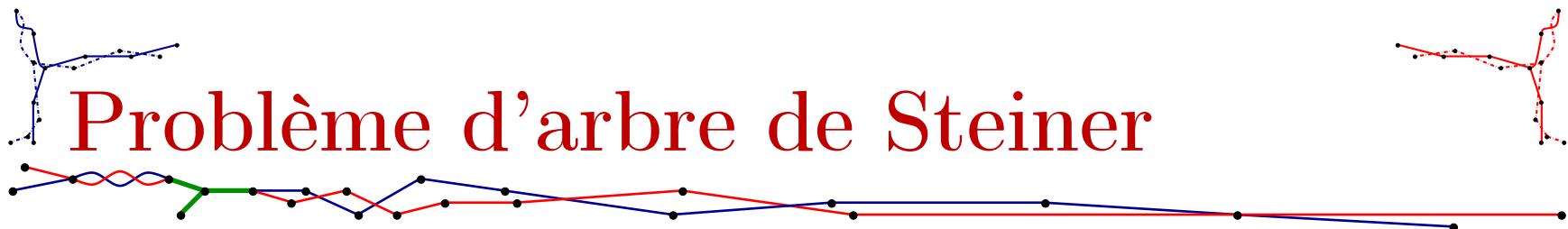
PRINCIPE : avec graphe G en input et arbre T en output :

$G \rightarrow$ liste d'arêtes de $G \rightarrow$ liste d'arêtes de $T \rightarrow$ hiérarchie

EXERCICE : liste d'arêtes de $T \rightarrow$ hiérarchie ?

TMBS.

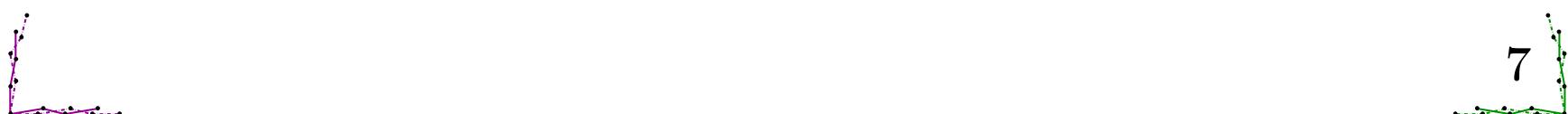




Problème d'arbre de Steiner

IN : Points, une liste de coordonnées de points en 2D

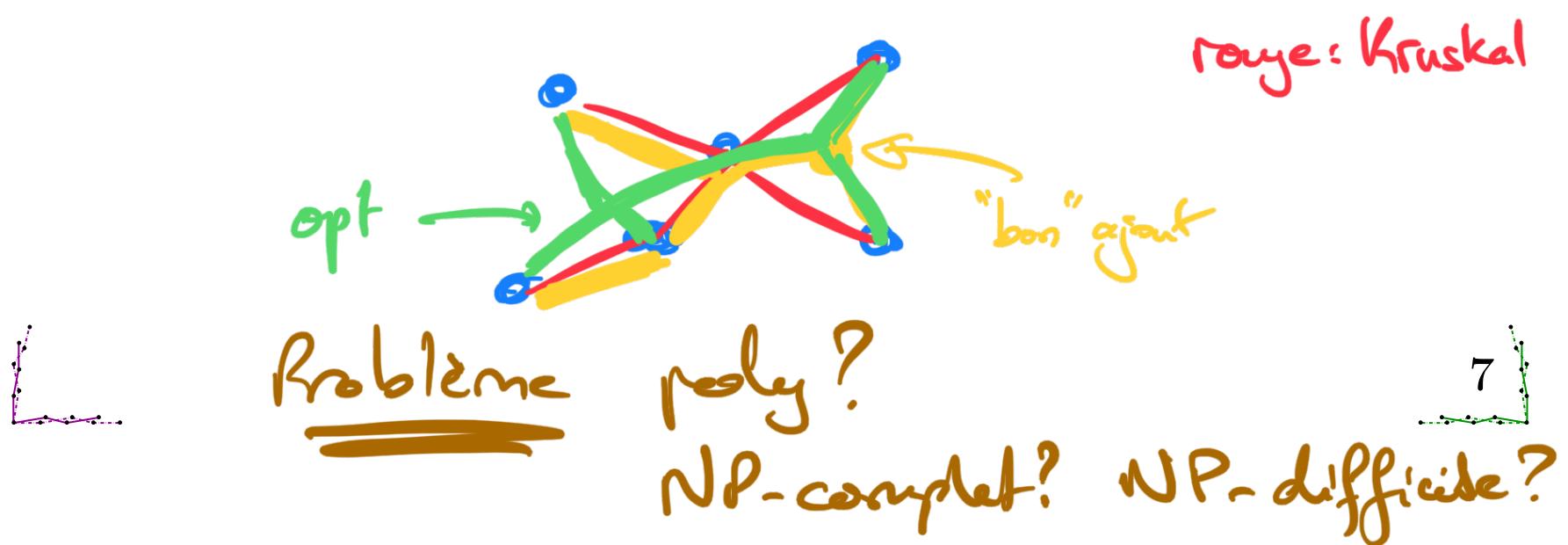
OUT : arbre Tree couvrant tous les points de la liste, de poids total minimum (somme de la distance entre toutes les arêtes).





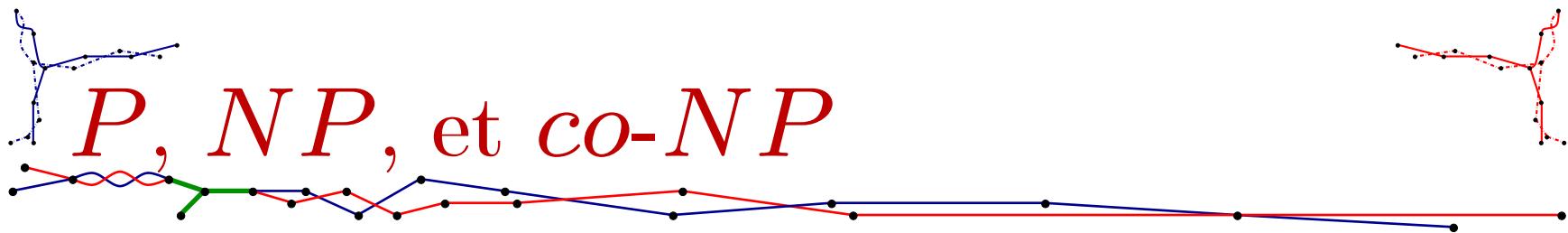
- ② IN : Points, une liste de coordonnées de points en 2D
- ③ OUT : arbre Tree couvrant tous les points de la liste, de poids total minimum (somme de la distance entre toutes les arêtes).

EXERCICE : différence avec arbre couvrant classique ?





Rappel express de la théorie de la complexité



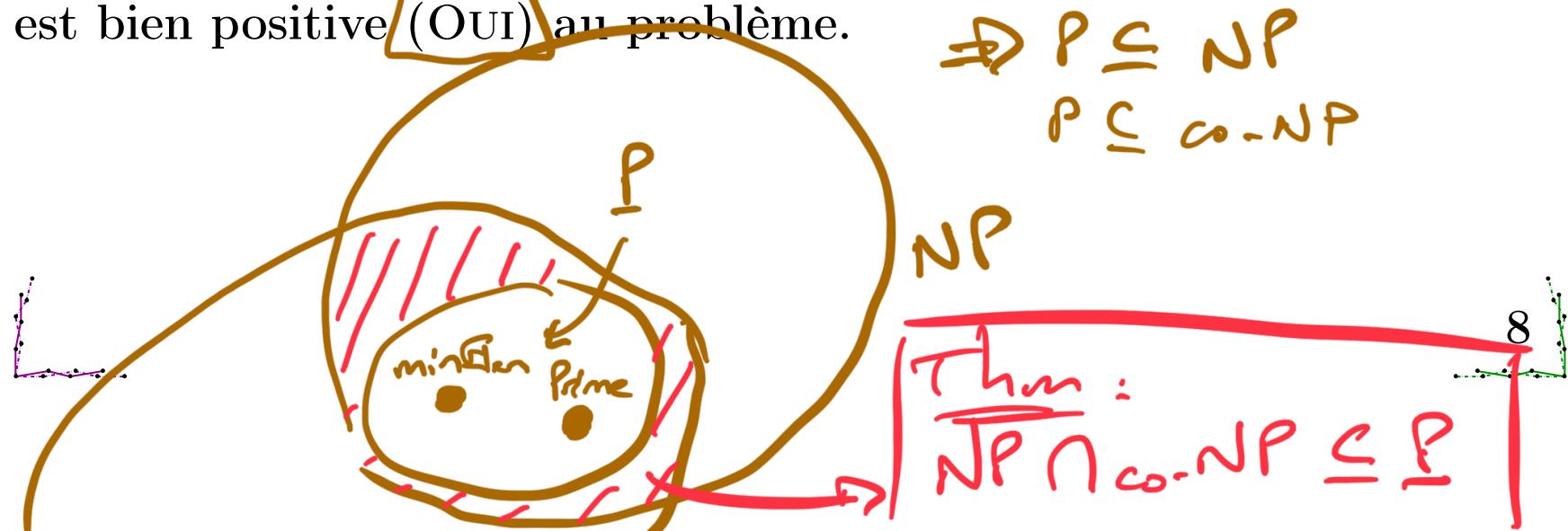
P , NP , et $co-NP$

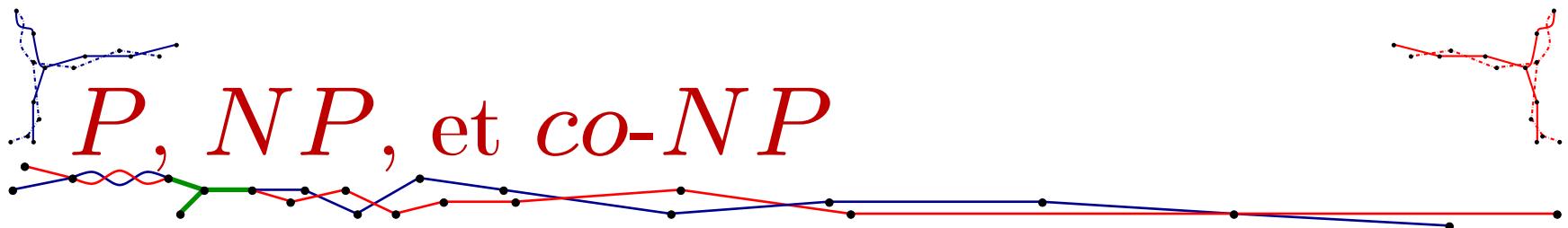
CLASSE P : l'ensemble de tous les problèmes de décision dont on peut calculer en temps polynomial en la taille de l'entrée (INPUT) et de la sortie (OUTPUT) la réponse (OUI/NON) au problème.

CLASSE NP : l'ensemble de tous les problèmes de décision dont on peut vérifier en temps polynomial en la taille de l'entrée (INPUT) et de la sortie (OUTPUT), moyennant un certain certificat, si la réponse est bien positive (OUI) au problème.

$$\Rightarrow P \subseteq NP$$

$$P \subseteq co-NP$$





P , NP , et $co-NP$

CLASSE P : l'ensemble de tous les problèmes de décision dont on peut calculer en temps polynomial en la taille de l'entrée (INPUT) et de la sortie (OUTPUT) la réponse (OUI/NON) au problème.

CLASSE $co-NP$: l'ensemble de tous les problèmes de décision dont on peut vérifier en temps polynomial en la taille de l'entrée (INPUT) et de la sortie (OUTPUT), moyennant un certain certificat, si la réponse est bien négative (NON) au problème.

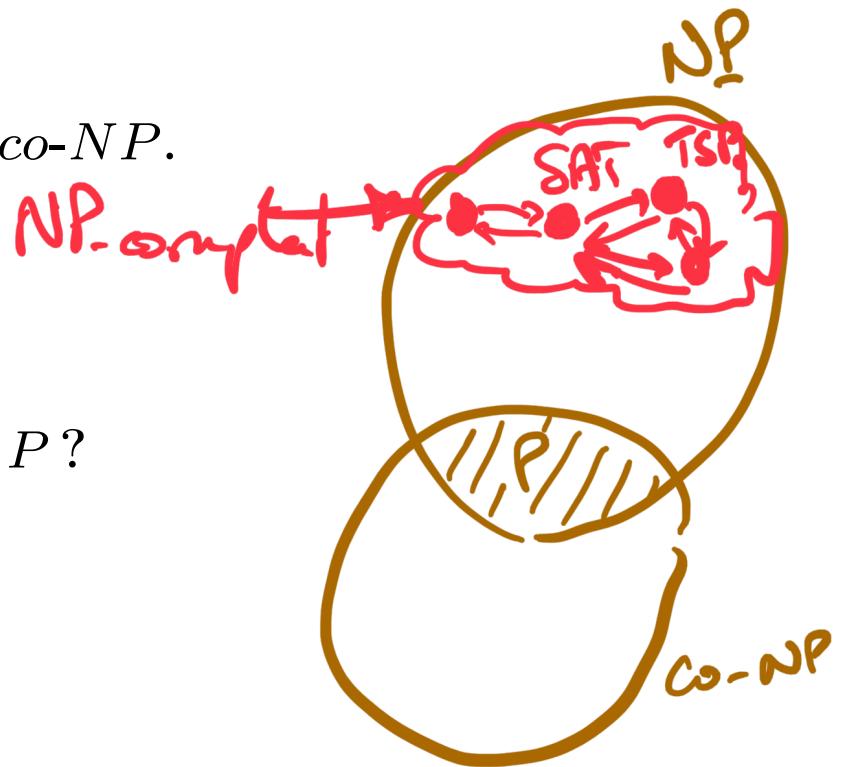


NP -complétude

PROPRIÉTÉ TRIVIALE : $P \subseteq NP \cap co-NP$.

THÉORÈME : $P = NP \cap co-NP$.

QUESTION : qu'y a-t-il dans $NP \setminus P$?



NP-complétude

PROPRIÉTÉ TRIVIALE : $P \subseteq NP \cap co\text{-}NP$.

THÉORÈME : $P = NP \cap co\text{-}NP$.

QUESTION : qu'y a-t-il dans $NP \setminus P$?

THÉORÈME (COOK) : Si $SAT \in P$ alors $NP \subseteq P$.

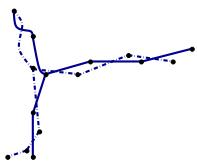
DÉFINITION : NP -complet = $\{Prob : Si\ Prob \in P\ alors\ NP \subseteq P\}$.

WANTED (1M\$) : $P = NP$ ou $P \neq NP$?

Thm: SteinerTreeDecision \in NP-complet

Consequence:

Si SteinerTreeDecision $\in P$ alors
 $P = NP$



Revenons à nos moutons (arbres)



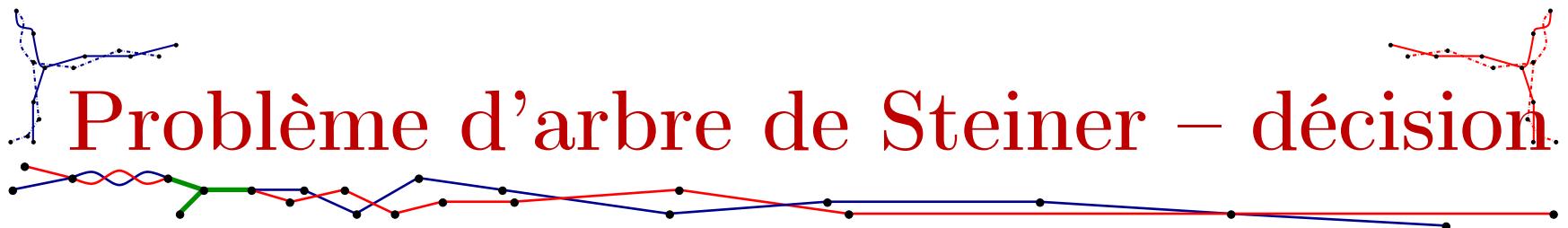


Problème d'arbre de Steiner – décision

IN : Points, une liste de coordonnées de points en 2D ; et B un réel.

OUT : existe-t-il un arbre Tree couvrant tous les points de la liste, de poids inférieur à B ?

EXERCICE : appartenance à NP ? a $co\text{-}NP$?



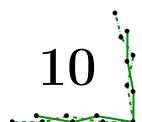
Problème d'arbre de Steiner – décision

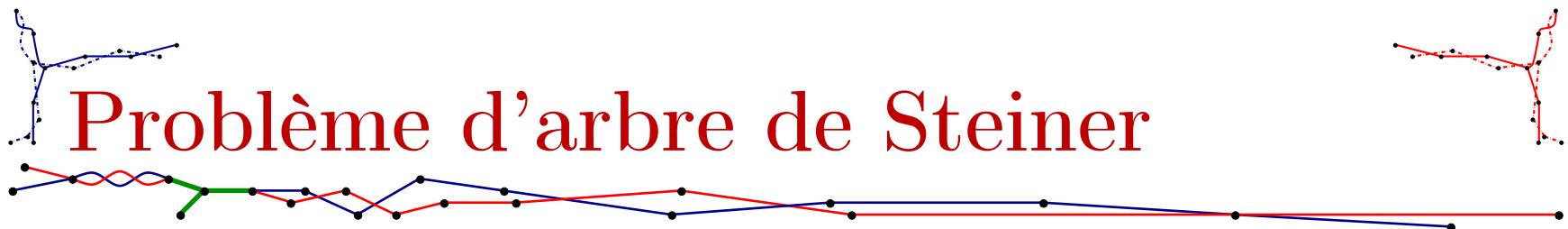
IN : Points, une liste de coordonnées de points en 2D ; et B un réel.

OUT : existe-t-il un arbre Tree couvrant tous les points de la liste, de poids inférieur à B ?

EXERCICE : appartenance à NP ? à $co\text{-}NP$?

THÉORÈME : *la version de décision du problème d'arbre de Steiner est NP-complet.*





Problème d'arbre de Steiner

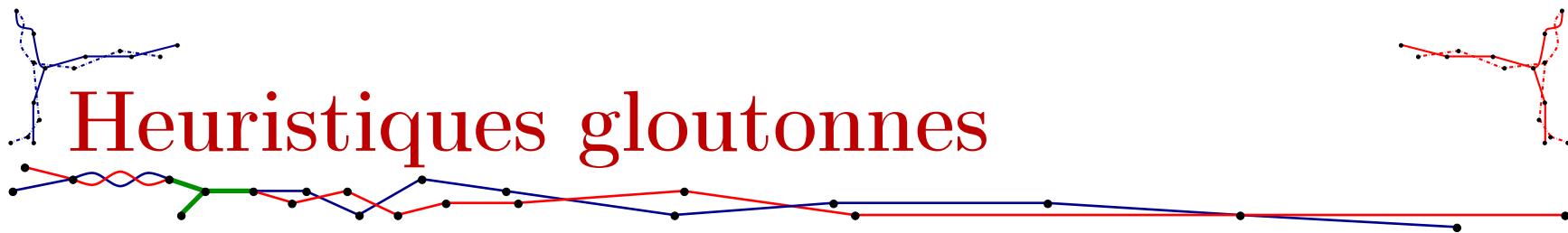
IN : Points, une liste de coordonnées de points en 2D

OUT : arbre Tree couvrant tous les points de la liste, de poids total minimum (somme de la distance entre toutes les arêtes).

EXERCICE : implémentation du test d'appartenance à NP ?

N.B. : problème de décision NP -complet \rightarrow problème d'optimisation associé est appelé NP -difficile

Rq: Arbre de Steiner est NP -difficile.
L^{parce que sa version de décision est NP -complet} 11

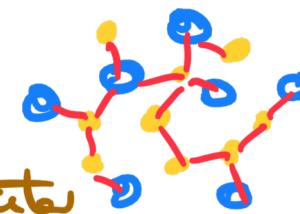


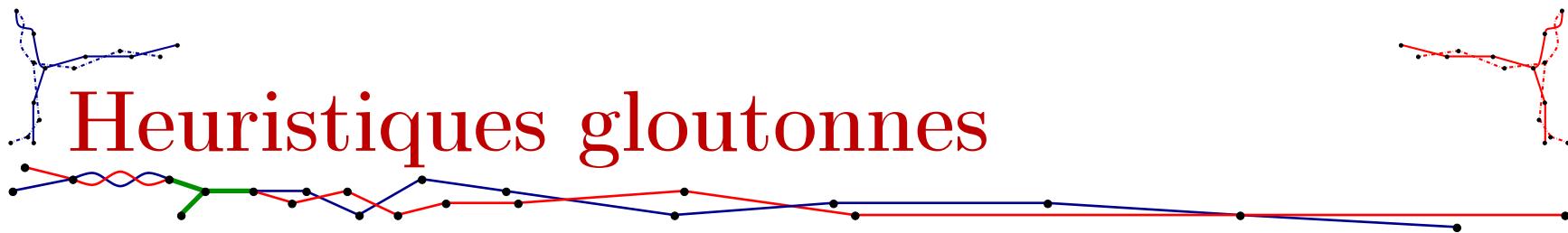
Heuristiques gloutonnes

PRINCIPE : ajouter des points à la liste Points, appeler Kruskal et comparer le résultat avec CurrentTree. Mettre à jour. Répéter...

NAÏF : essayer tous les coordonnées possibles.

Q: comment déterminer les
points (en jaune) à ajouter
dans Input?





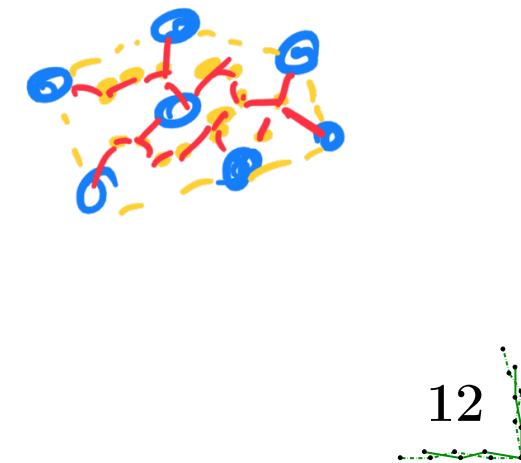
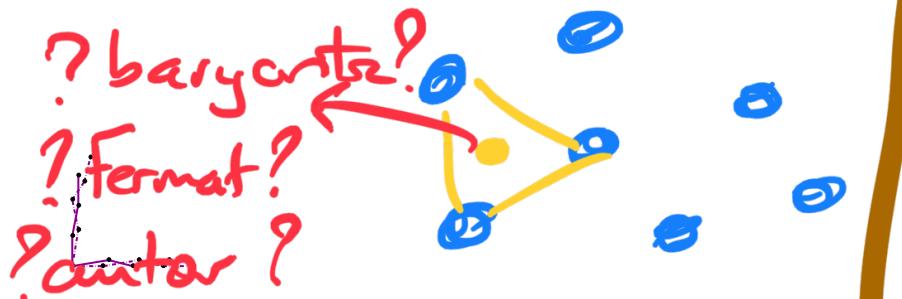
Heuristiques gloutonnes

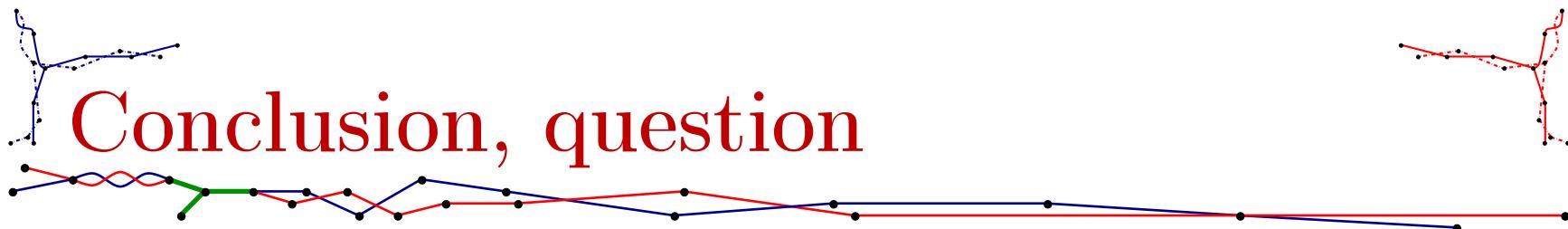
PRINCIPE : ajouter des points à la liste Points, appeler Kruskal et comparer le résultat avec CurrentTree. Mettre à jour. Répéter...

NAÏF : essayer tous les coordonnées possibles.

NAÏF AMÉLIORÉ : pour tout triangle qui ne contient aucun autre point, essayer tous les coordonnées possibles.

EXERCICE : mieux ?





Conclusion, question

CONCLUSION :

- ARBRECOUVRANTMIN algorithme Kruskal en $O(m \log n)$
- ARBRESTEINER NP-difficile
 - heuristique : recherche locale
 - optimisation locale : à l'aveugle, barycentre Torricelli-Fermat

QUESTION :

- implantation ? (voir TME)

énoncé $m \log n + \alpha$ \rightarrow ~~α~~



