Le Voyageur de Commerce

Année 2022 – 2023

Philippe Aumaitre, Romain Dhenry, Zoé Surelle, élèves de classe de Terminale générale

Établissement(s) : Lycée Raynouard Brignoles

Enseignant·e(s) : Denis Guicheteau, Nelly Mourau, Marc Brunet

Chercheur·Chercheuse(s) : Frédéric Havet, INRIA, Thierry Champion, Université de Toulon.

1. Présentation du sujet

Un voyageur de commerce souhaite organiser sa tournée de façon optimale. Ainsi, il souhaiterait parcourir toutes ces villes en optimisant le trajet, c'est-à-dire en minimisant la distance parcourue lors de ce trajet. Pour ce faire, il n'est limité ni par le point de départ, ni par le point d'arrivée, il doit simplement passer par toutes les villes.
On peut donc se demander : quel est le chemin le plus court ?

2. Résultats

Pour un petit nombre de points, on pourra déterminer le plus court chemin à l'aide d'un algorithme qui teste tous les chemins possibles dans un temps factoriel. Ensuite, pour un grand nombre de points, on a développé notre propre algorithme qui permet de calculer dans un temps polynomial un chemin court mais qui n'est pas forcément le plus court.

3. Les différentes démarches

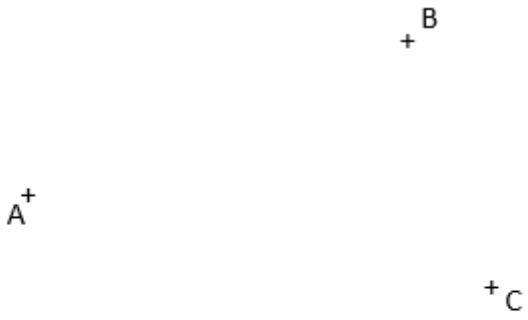
3.1. L'idée dite "naïve"

Dans un premier temps, l'idée la plus spontanée nous est venue à l'esprit : à chaque intersection, on prend le chemin le plus court sans revenir en arrière.

Pour 2 villes :

Le plus court est évidemment de les joindre par un segment

Pour 3 villes :



$$BC < AB < AC$$

Chemin minimal : AB + BC ou CB + BA

Dans le cas de 3 villes, le chemin minimum s'obtient en choisissant les deux plus courts chemins successivement.

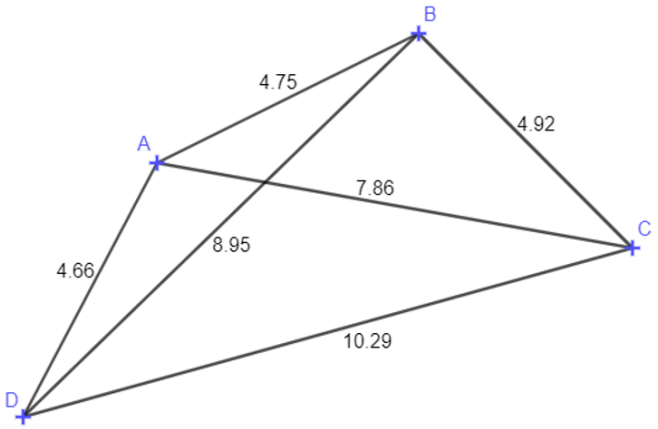
Propriété :

S'il y a trois villes A, B et C telles que $AB < BC < CA$
La longueur minimale est la longueur du chemin : AB + BC

Pour plus de trois villes :

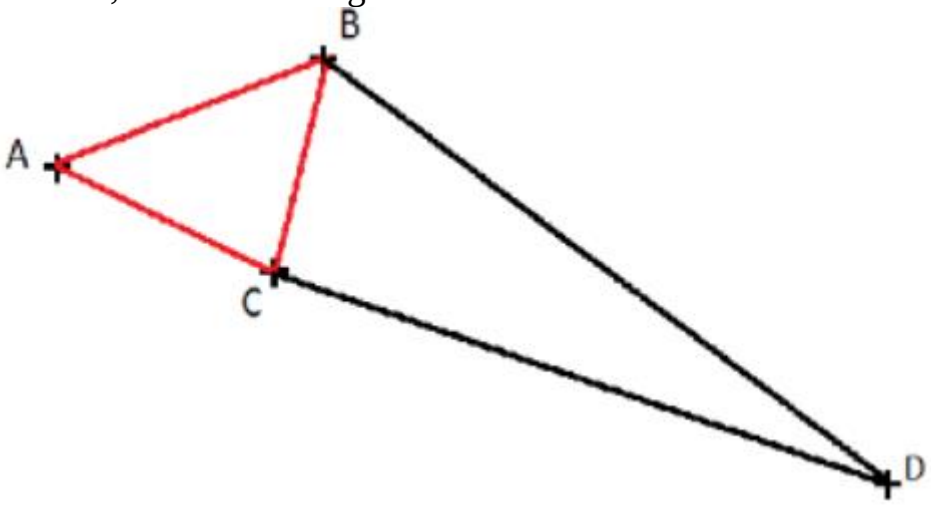
On tente de prendre le chemin le plus court à chaque nœud, en évitant de retourner en arrière.

Exemple pour 4 villes :



Suivant le point de départ, nous obtenons des résultats différents :
Trajet ADBC : 18,53
Trajet BACD : 19,7
Trajet CBAD : 14,33
Trajet DABC : 14,33

Autre problème, dans une configuration comme celle-ci :



L'algorithme ne parvient pas à choisir un chemin vers D car trop loin des autres.

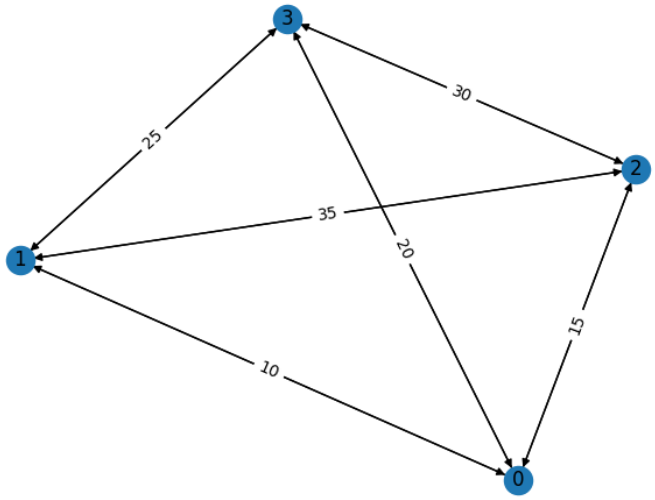
Conclusion : il faut changer d'idée.

3.2. L'utilisation de matrice :

Nous avons naturellement présenté les chemins sous forme de graphe, mais cette représentation ne nous permettait pas de faire des calculs rapidement.
Ainsi nous avons changé et nous sommes passés à une représentation sous forme de matrice.
Par exemple :

[0, 10, 15, 20],
[10, 0, 35, 25],
[15, 35, 0, 30],
[20, 25, 30, 0]

correspondait à ce graphe ci :



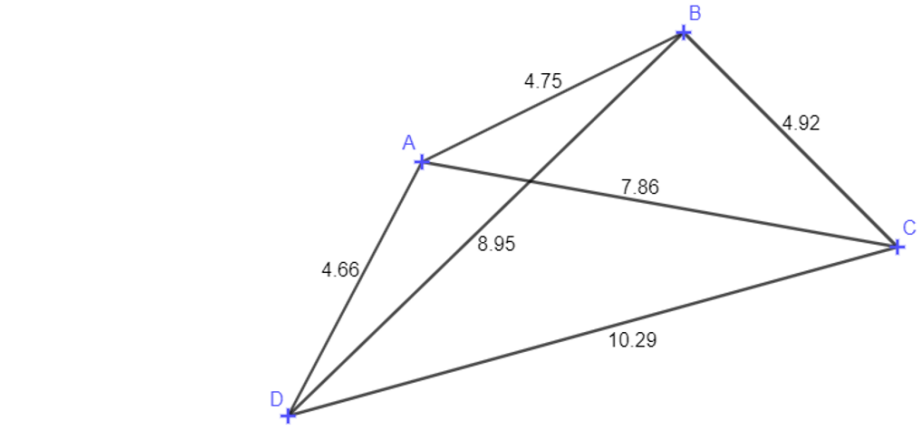
De plus, nous avons aussi créé un programme qui prend en paramètre des coordonnées de points et qui renvoie la matrice correspondant en calculant les distances entre chaque point.

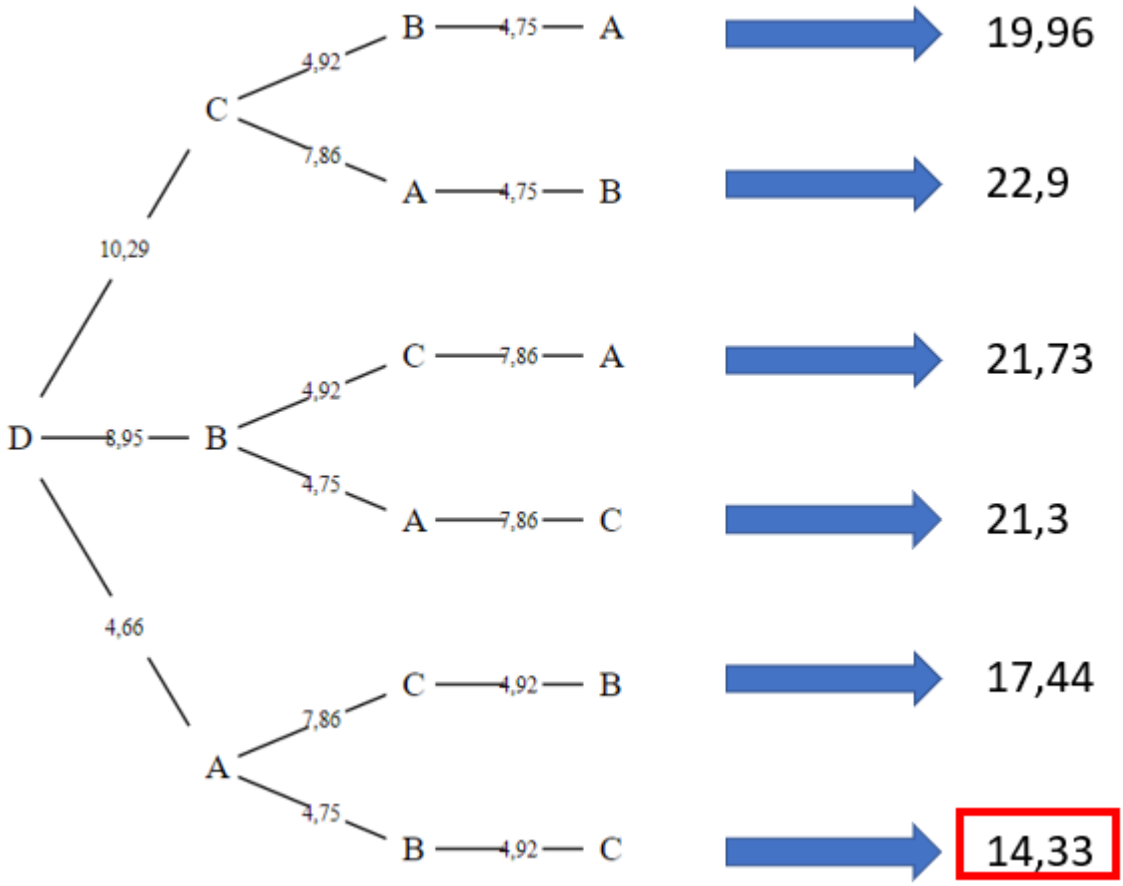
```
#fonction qui prend 7 points et construit la matrice de tous les chemins possibles
def graph_routes():
  list = [0] * 7
  dist = np.zeros((7, 7))
  for i in range(7):
     x = input('Points : ').split(' ')
     list[i] = x

  for i in range(7):
     a = list[i]
     for j in range(7):
        b = list[j]
        dist[i][j]= round(sqrt((int(a[0])-int(b[0]))**2+(int(a[1])-int(b[1]))**2),2)
  return dist
```

3.2. L’algorithme de force brute.

Dans un second temps, une idée plutôt logique s'est imposée à nous : on calcule

tous les chemins possibles. Cette démarche est réalisable à la main à l'aide d'un arbre pondéré.





Exemple d'arbre pour un départ de D.

Pour 4 points, il a fallu faire 24 calculs pour trouver tous les chemins possibles. Pour *n* points, il y a *n* factoriel possibilités.

QU'EST-CE QUE LA FONCTION "FACTORIEL" ?
La fonction factoriel de *n* (notée *"n!"*), correspond au produit des nombres de 1 à *n* (par exemple 4! = 1 x 2 x 3 x 4). Cette notion est utilisée dans le cas des combinatoires, plus précisément des permutations.
Une permutation, c'est lorsque j'ai par exemple 10 livres dans une bibliothèque, et que je souhaite les ranger tous aléatoirement : j'aurais alors 10! rangements de livres différents possibles.

Cela signifie que le nombre de calculs peut vite devenir extrêmement grand, et donc long à calculer. Pour vous donner un ordre d'idée, l'ordinateur le plus puissant du monde peut effectuer environ 10^{15} calculs à la seconde, or 20 factoriel vaut environ 10^{18}, et il y a bien plus de 20 villes rien qu'en France.

Conclusion : Il faut changer d'idée. Mais pour de petits nombres, nous allons nous servir de cet algorithme comme base pour comparer nos futures idées.

Programme python de force brute :

```
import itertools

def tsp(graph, start):
  # obtenir tous les chemins possibles à partir du noeud de départ
  paths = list(itertools.permutations(range(len(graph))))

  # initialiser la distance minimale à l'infini
  min_distance = float('inf')
  min_path = []

  # pour chaque chemin
  for path in paths:
    # initialiser la distance totale à 0
    total_distance = 0

    # pour chaque noeud dans le chemin
    for i in range(len(path) - 1):
      # ajouter la distance entre le noeud actuel et le prochain
      total_distance += graph[path[i]][path[i+1]]

    # si la distance totale est inférieure à la distance minimale actuelle
    if total_distance < min_distance:
      # mettre à jour la distance minimale et le chemin minimal
      min_distance = total_distance
```

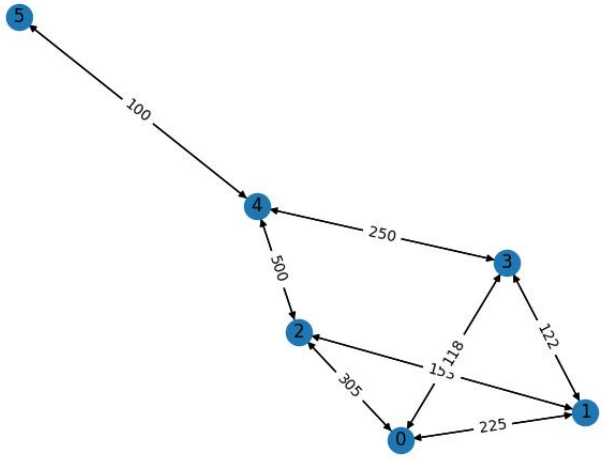
```
    min_path = path

  # renvoyer la distance minimale et le chemin minimal
  return min_distance, min_path

# exemple d'utilisation
graph = [[0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]]

min_distance, min_path = tsp(graph, 0)
print(min_distance)  # 50
print(min_path)  # (2, 0, 1, 3)
```

3.3. L'idée finale

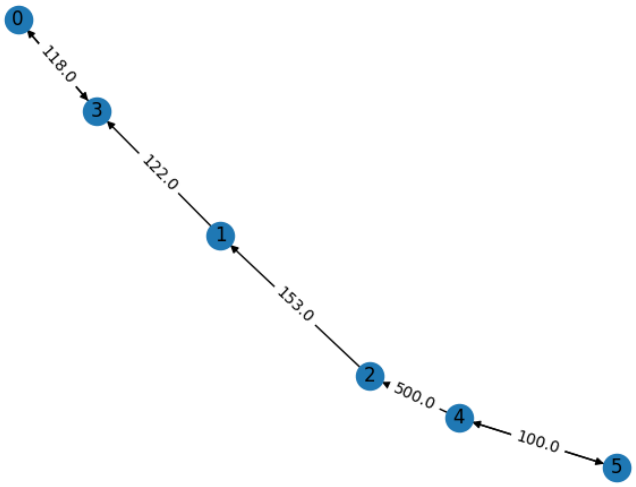
Enfin, l'idée sur laquelle nous nous sommes concentrés correspond à la démarche suivante :

- Nous relions des grappes de points les plus proches (avec la méthode naïve).
- Ensuite nous relions les extrémités des grappes.

Par exemple, une configuration comme celle-ci :



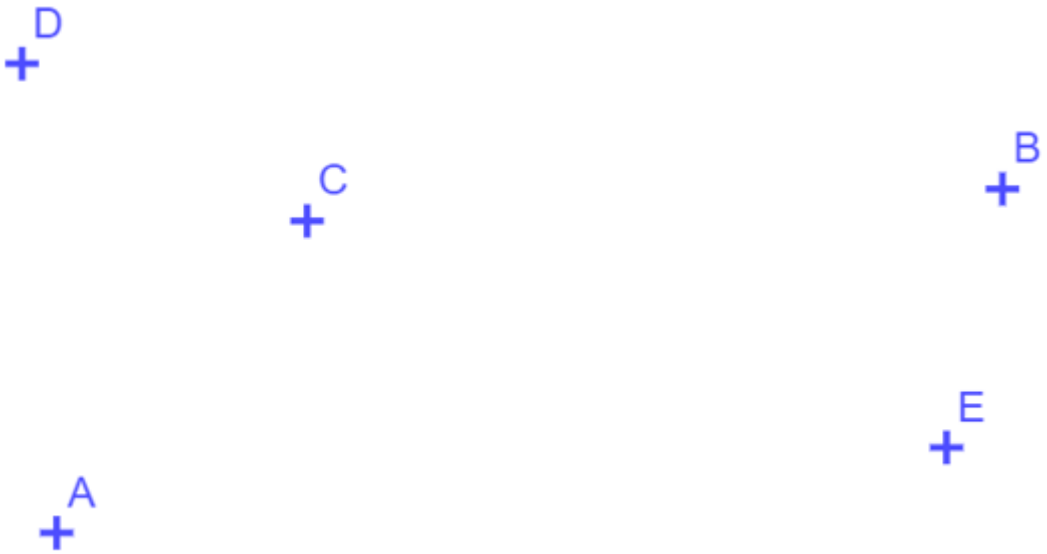
Donnera une grappe comme celle-ci :



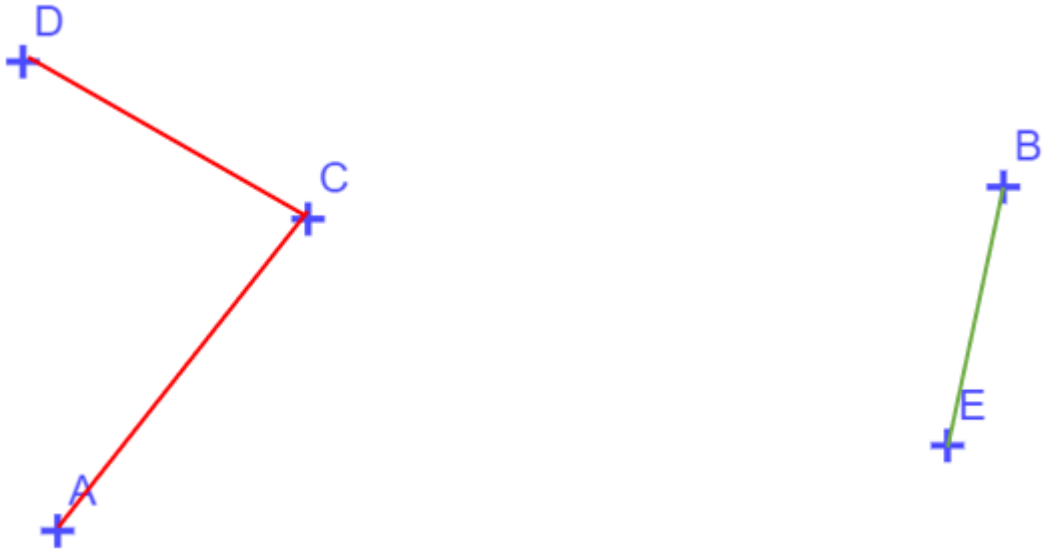
Cependant, cette idée nous donne un chemin court, mais pas forcément le plus court.
Ainsi, il faut vérifier que notre solution est cohérente, nous utilisons en premier lieu l'algorithme de force brute pour comparer notre solution avec la solution optimale.
Malheureusement, si le nombre de points devient grand, il est difficile de vérifier si notre solution donne bien un chemin suffisamment court.
De là, nous avons décidé de mettre en place une borne inférieure, qui nous permet d'avoir une idée de la longueur finale du chemin que nous renvoie notre algorithme.

LA BORNE INFERIEURE : LE CALCUL
Comme dit ci-dessus, on souhaite calculer une valeur basse pouvant nous donner une idée de la longueur minimale du chemin le plus court.
Ainsi, pour relier nos n points, il faut $(n - 1)$ segments.
On va prendre pour chaque point, la longueur la plus petite, on additionne ces n longueur. Puis on multiplie par $\frac{n-1}{n}$ pour obtenir une valeur approchée de la longueur des $(n-1)$ segments.

Fonctionnement de l'algorithme sur un exemple :

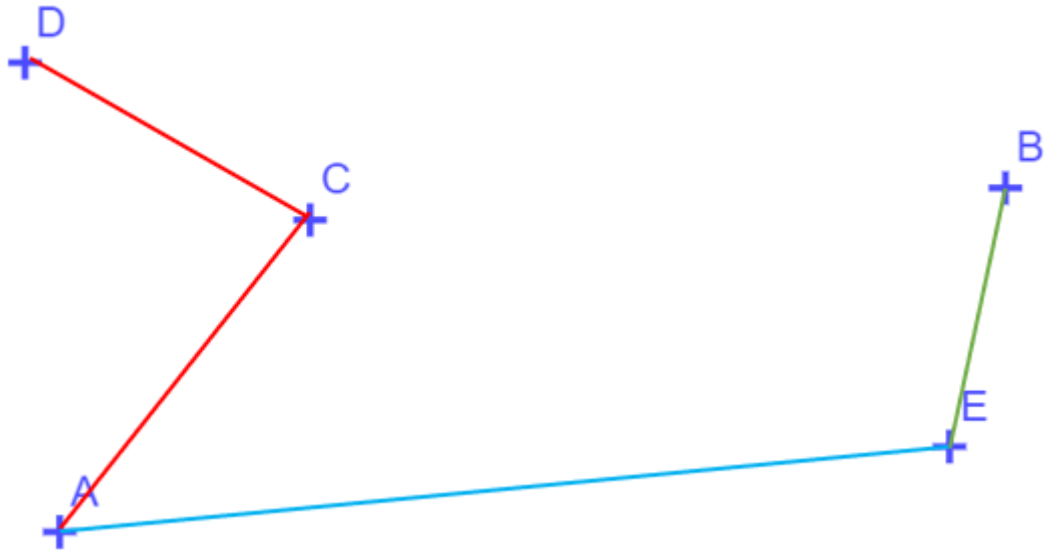


1ere étape : on relie les points les plus proches en utilisant la matrice.

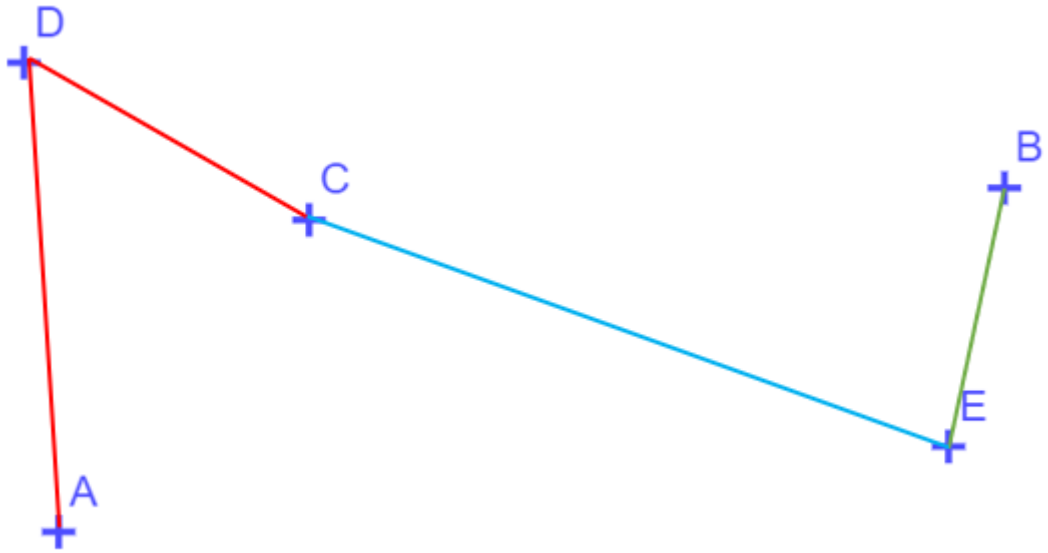


Ainsi, A est relié à C, B est relié à E, C est relié à D, D est relié à C et E est relié à B.

2ème étape : on regarde parmi les extrémités des grappes, le chemin le plus court pour les relier



On peut voir que dans ce cas, le chemin n'est pas optimal car le chemin suivant est meilleur :



C'est le chemin que nous obtenons par force-brute.
Néanmoins, notre algorithme nécessite dans le meilleur des cas *n* calculs. Nous n'avons pas réussi à déterminer le nombre de calculs pour le pire des cas.

Voici un lien vers notre programme : https://trinket.io/python3/507ab39819

4. Conclusion

Le problème du voyageur de commerce est en soi facile à résoudre, mais c'est le temps de résolution qui pose problème.

En effet, la raison pour laquelle le voyageur de commerce fait partie des problèmes du millénaire, c'est parce qu'on ne sait pas si on peut le résoudre dans

un temps polynomial. Pour le dire autrement, le temps de calculs augmente d'une manière bien trop considérable lorsque qu'on augmente le nombre de points

QU'EST-CE QU'UN PROBLEME DU MILLENAIRE ?
En l'an 2000, 7 problèmes ont été introduits par l'institut de mathématiques Clay et dont la résolution apporte la maudique somme d'un million de dollars. Ce sont tous des problèmes extrêmement difficiles à résoudre de manière précise et nécessite des techniques mathématiques très avancées pour même s'approcher d'une solution à peine précise. Pour le moment, seule la conjecture de Poincaré a été résolue par Grigori Perelman, un mathématicien russe.