

Rapport du TP n°= 4

Théorie de programmation et graphes orientés

Thème :

Problème du PVC (Le problème du voyageur de commerce)

Réalisé par :

- BOUTIBA Karim
- REMIL Youcef
- ✓ **Option : SIQ**
- ✓ **Groupe : 2**

Proposé par :

- Mr HADIM Boukhalfa

Promotion : 2018/2019

1. Introduction :

1.1 Enoncé du problème :

Le problème du voyageur de commerce, étudié depuis le 19e siècle, est l'un des plus connus dans le domaine de la recherche opérationnelle. C'est déjà sous forme de jeu que William Rowan Hamilton a posé pour la première fois ce problème, dès 1859. Sous sa forme la plus classique, son énoncé est le suivant : « Un voyageur de commerce doit visiter une et une seule fois un nombre fini de villes et revenir à son point d'origine. Trouvez l'ordre de visite des villes qui minimise la distance totale parcourue par le voyageur ». Ce problème d'optimisation combinatoire appartient à la classe des problèmes **NP-Complets**.

1.2 Domaines d'applications :

Les domaines d'application sont nombreux : problèmes de logistique, de transport aussi bien de marchandises que de personnes, et plus largement toutes sortes de problèmes d'ordonnancement. Certains problèmes rencontrés dans l'industrie se modélisent sous la forme d'un problème de voyageur de commerce, comme l'optimisation de trajectoires de machines-outils : comment percer plusieurs points sur une carte électronique le plus vite possible ?

2. Représentation du problème :

Le problème commerce peut être modélisé à l'aide d'un graphe constitué d'un ensemble de sommets et d'un ensemble d'arêtes. Chaque sommet représente une ville, une arête symbolise le passage d'une ville à une autre, et on lui associe un poids pouvant représenter une distance, un temps de parcours ou encore un coût.

Voici un exemple de graphe à 4 sommets :

Résoudre le problème du voyageur de commerce revient à trouver dans ce graphe un cycle passant par tous les sommets une unique fois (un tel cycle est dit « hamiltonien ») et qui soit de longueur minimale. Pour le graphe ci-contre, une solution à ce problème serait le cycle 1, 2, 3, 4 et 1, correspondant à une distance totale de 23. Cette solution est optimale, il n'en existe pas de meilleure.

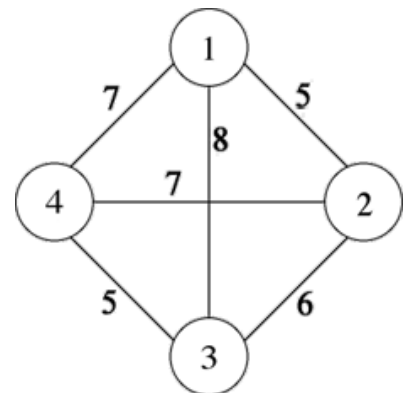


Figure 1 : Exemple de graphe complet à 4 sommets.

Comme il existe une arête entre chaque paire de sommets, on dit que ce graphe est « complet ». Pour tout graphe, une matrice de poids peut être établie. En lignes figurent les sommets d'origine des arêtes et en colonnes les sommets de destination ; le poids sur chaque arête apparaît à l'intersection de la ligne et de la colonne correspondantes. Pour notre exemple, cette matrice est la suivante :

	0	5	8	7
	5	0	6	7
	8	6	0	5
	7	7	5	0

3. Méthodes de résolution :

Comme pour le problème du sac à dos (un autre des problèmes les plus connus dans le domaine de l'optimisation combinatoire et de la recherche opérationnelle), il existe deux grandes catégories de méthodes de résolution : les méthodes exactes et les méthodes approchées. Les méthodes exactes permettent d'obtenir une solution optimale à chaque fois, mais le temps de calcul peut être long si le problème est compliqué à résoudre. Les méthodes approchées, encore appelées heuristiques, permettent quant à elles d'obtenir rapidement une solution approchée, mais qui n'est donc pas toujours optimale.

3.1 Méthodes exactes :

Pour le problème du voyageur de commerce, l'une des méthodes exactes les plus classiques et les plus performantes reste la Procédure par Séparation et Evaluation (PSE). Cette méthode repose sur le parcours d'un arbre de recherche. Dans un chemin de cet arbre, le premier nœud représente la ville de départ, son successeur la deuxième ville visitée, puis la troisième ville visitée, etc. À chaque étape de l'algorithme, on crée autant de nœuds qu'il reste de villes à visiter. À chaque nœud, le choix consiste à sélectionner la prochaine ville à visiter parmi les villes restantes. Ainsi, voici l'arbre de recherche pour l'exemple présenté ci-dessus :

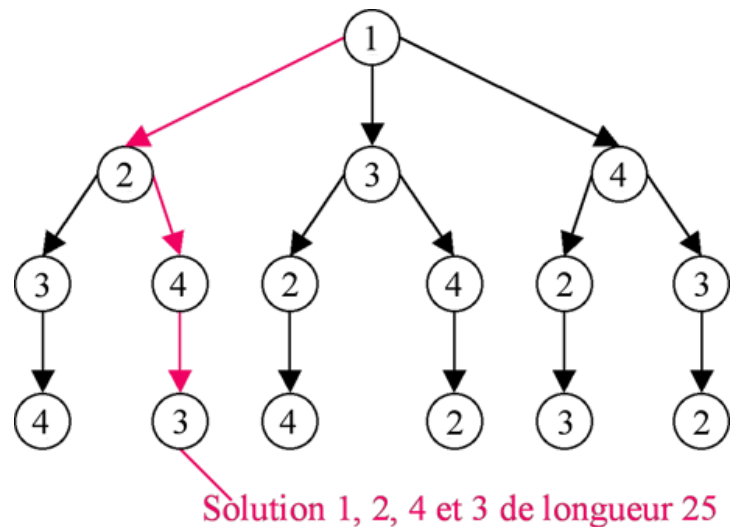


Figure 2 : Arbre de recherche associé au graphe

✓ L'Algorithme de parcourt BFS pour la solution triviale (exacte) :

- On définit une structure de graphe commune à toutes les solutions :

```
Structure {  
    ns : entier ;    /* nombre de sommets */  
    mat_adj [] [] : tableau d'entier init (0) ;  
} Graphe ;
```

- On définit une structure de chemin commune à toutes les solutions :

```
Structure {  
    ns : entier ;    /* nombre de sommets*/  
    cout : entier ;  
    chem : tab [n] d'entier ;  
} Chemin ;
```

- On définit les variables globales :

```
cycle_opt : Chemin ;  
cout_opt : init (INFINI) ;
```

- On utilise le parcourt BFS itératif en utilisant une file de chemin :

```
Procedure BFS ( g : Graphe , depart : entier )  
{  
    ch, nouv : Chemin ;  
    n = g.ns ;  
    ch.ns = 1 ;  
    ch.chem [0] = depart  
    ch.cout = 0 ;  
    CreerFile (File) ;  
    Enfiler (File, ch) ;  
    TANT QUE ( Non FileVide (File))  
    {  
        Defiler (File, ch) ;  
        SI (ch.ns < n )  
        {  
            dernier = ch.chem [ch.ns - 1] ;  
            POUR s : 0 -> (n-1) faire
```

```

    {
        SI ( s n'existe pas dans ch )
        {
            Copier ch dans nouv
            nouv.chem [ch.ns] = s ;
            nouv.ns ++ ;
            nouv.cout += g.mat_adj [dernier][s]
            Enfiler(File,nouv) ;
        }
    }
}
SINON      /* ch.ns == n */
{
    SI (ch.cout < cout_opt)
    {
        Copier ch dans cycle_opt
        cout_opt = ch.cout
    }
}
}
}

```

3.2 Méthodes approchées :

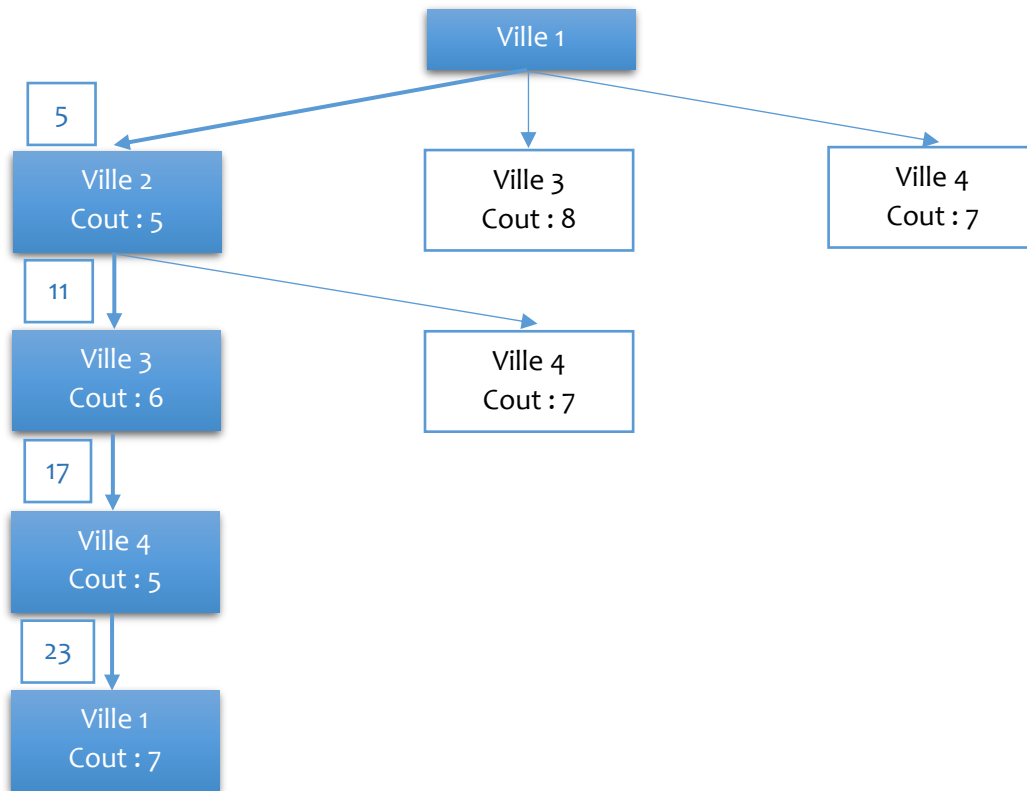
Dans le cas d'un nombre de villes si grand que même les meilleures méthodes exactes nécessitent un temps beaucoup trop long de résolution, des méthodes approchées, ou algorithmes d'approximation, sont utilisées. Elles permettent d'obtenir en un temps très rapide de bonnes solutions, pas nécessairement optimales, mais d'une qualité suffisante.

Il existe de nombreux algorithmes d'approximation. Chacun a ses avantages et ses inconvénients, Une famille de méthodes est celle des algorithmes gloutons, dont les uns des plus connus sont :

3.2.1 L'algorithme du plus proche voisin :

C'est un algorithme très simple : La première étape repose sur le choix aléatoire d'une première ville, et les étapes suivantes consistent à se déplacer de ville en ville en appliquant la règle du plus proche voisin, c'est-à-dire en sélectionnant la prochaine ville telle que le poids entre la ville courante et la prochaine ville soit minimal, et ce, jusqu'à avoir visité toutes les villes. Il faut enfin revenir à la première ville choisie, pour obtenir un cycle.

- Considérons l'exemple du graphe précédent (Figure 1) et déroulons l'algorithme :
 - Supposons que la ville de départ et la ville 1



- **Le chemin est : 1 -> 2 -> 3 -> 4 -> 1**

✓ L'Algorithme de Gluton (Plus proche voisin) :

```

Procure GlutonVoisin ( g : Graphe , depart : entier )
{
    n = g.ns ;
    existe : tableau [n] de booleen init (Faux) ;
    existe [depart] = faux ;

    cycle_opt.chem[0] = depart ;
    cycle_opt.ns = 1 ;

    j = depart ;
    s = depart ;

    POUR i : 1 -> (n-1) faire
    {
        k = 0
    }
  }

```

```

    TANT QUE ( existe [k] ) faire k ++ ;
    inter = g.mat_adj [j][k] ;
    s = k ;
    k ++ ;

    TANT QUE ( k < n )
    {
        SI ( Non existe[k] ET g.mat[j][k] < inter )
        {
            s = k ;
            inter = g.mat[j][k] ;
        }
        k ++
    }

    j = s ;
    existe[s] = vrai ;
    cycle_opt.chem[i] = s ;
    cycle_opt.cout += inter ;
    cycle_opt.ns ++ ;
}
cycle_opt.chem[n] = depart ;
cycle_opt.cout += g.mat_adj [s][depart] ;
cycle_opt.ns ++
}

```

3.2.2 L'algorithme de Kruskal (trie d'arêtes) :

L'idée de l'algorithme est aussi simple, elle consiste à trier les arêtes par ordre croissant (Il existe C_n^2), puis prendre les arêtes une à une dans l'ordre en considérant les deux conditions suivantes :

- Aucun sommet ne doit avoir un degré > 2 .
- Le seul cycle formé est le cycle final, quand le nombre d'arêtes acceptées est égal au nombre de sommets du graphe.

- Considérons toujours l'exemple de la figure 1 :
Les arêtes triées selon leurs couts :

Ville source	1	3	2	2	1	1
Ville destination	2	4	3	4	4	3
Cout	5	5	6	7	7	8
Possibilité d'insertion	Oui	Oui	Oui	Non	Oui	Non

- Le chemin est : 1 -> 2 -> 3 -> 4 -> 1

✓ L'Algorithme de Gluton (Kruskal) :

- On définit une structure de chemin commune à toutes les solutions :

```
Structure {
    source : entier ;
    destination : entier ;
    cout : entier ;
} Arete ;
```

- L'algorithme de Kruskal est donné alors comme suit :

```
Procureur GlutonKruskal ( g : Graphe , depart : entier )
{
    n = g.ns ;
    nb_aretes = n (n-1) / 2 ;
    aretes = tableau [nb_aretes] d'arete ;
    cpt = 0 ;

    POUR i : 0 -> (n-2)
    {
        POUR j : i+1 -> (n-1)
        {
            aretes [cpt] = creerArete (i , j , g.mat_adj [i][j]) ;
            cpt ++ ;
        }
    }

    Trier (aretes) ;          /* selon ordre décroissant */
    tmp = 0 ;
    i = 0 ;
    grapheReduit : Graphe init (NULL) ;
    degree : tableau [n] d'entier init (0) ;

    TANT QUE ( i < nb_aretes ET tmp < n )
    {
        Ajouter (grapheReduit,arete[i]) ;
```



```

SI (Non ExisteCycle (grapheReduit) ET degree [arete[i].source] < 2
  ET degree [arete[i].destination] < 2)
{
    tmp ++ ;
    degree [arete[i].source] ++ ;
    degree [arete[i].destination] ++ ;
}
SINON
{
    Supprimer (grapheReduit,arete[i]) ;
}
i ++ ;
}
}

```

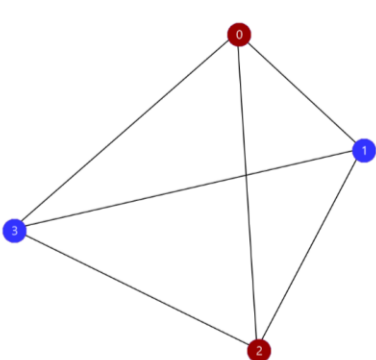
- Maintenant qu'on a le graphe réduit qui contient que les arêtes prises on peut tirer facilement le cycle optimal ainsi que son coût ;

4. Présentation de notre application :

- **Plateforme** : Desktop
- **Langage de programmation** : Java / JavaFX

TP4 : Problème du voyageur du commerce

Entrez le nombre de sommets : Sommet de départ :



Solution de Gluton (Plus proche voisin) TE : 1 ms Cout : 23 Chemin : 0 1 2 3 0
Solution de Gluton (Trie des arêtes) TE : 1 ms Cout : 23 Chemin : 0 1 2 3 0
Solution Exacte ou Triviale TE : 1 ms Cout : 23 Chemin : 0 1 2 3 0

© Réalisé par Boutiba Karim & Remil Youcef 2018-2019

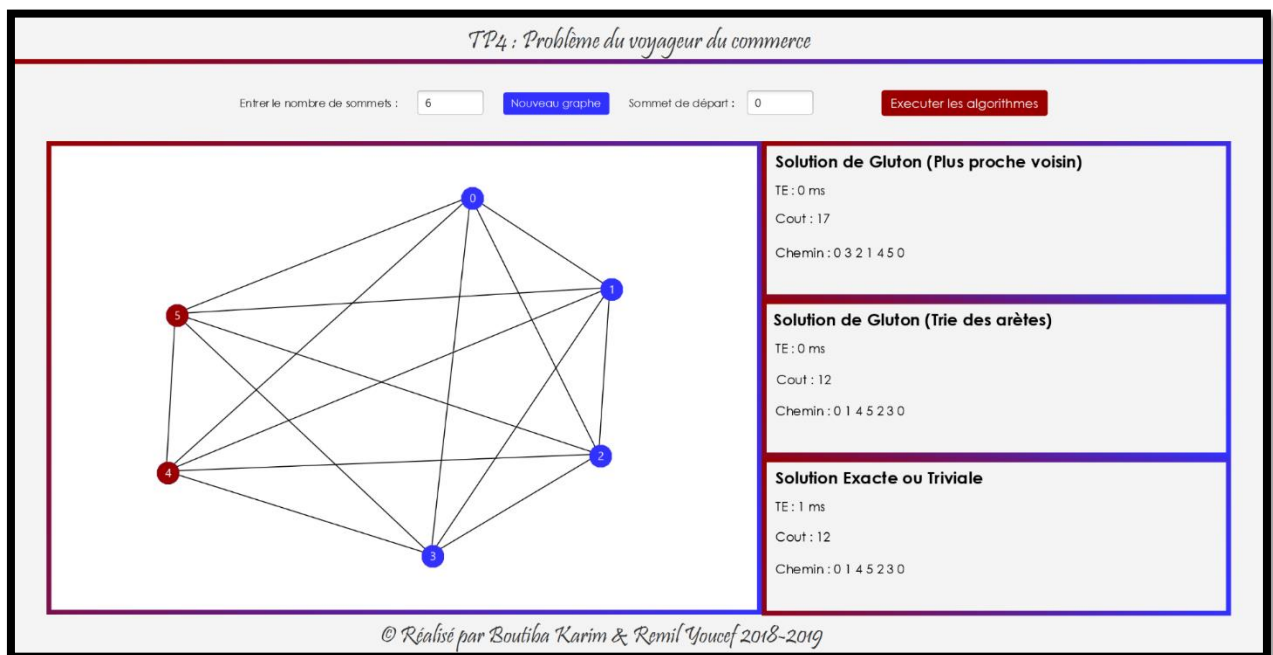
5. Tests et comparaison :

5.1 Tests d'efficacité (Evaluation selon le cout optimale) :

- Prenons l'exemple suivant :
Soit un graphe dont la matrice d'adjascence est la suivante :

0	2	5	1	4	10
	0	2	2	2	7
		0	1	2	5
			0	3	6
				0	1
					0

- L'exécution des 3 algorithmes donne les résultats suivants :



- **Interprétation :**

Pour cette exemple la méthode de Gluton (plus proche voisin) ne donne pas le cout optimal tandis que la solution de Gluton de Kruskal donne un cout optimal égal à celui trouvé par la méthode exacte mais ce n'est pas

toujours le cas , en résumé les solutions de Gluton donnent des résultats approchés mais qui sont bonnes en un temps réduit.

5.2 Tests de rapidité (Evaluation selon le temps d'exécution) :

Nombres de villes	Temps d'exécution (Plus proche voisin)	Temps d'exécution (Kruskal)	Temps d'exécution (Méthode exacte)
4	0 ms	2 ms	0 ms
8	1 ms	3 ms	13 ms
9	1 ms	3 ms	266 ms
10	1 ms	4 ms	26,458 s

Interprétation :

- Le temps d'exécution accroit exponentiellement pour la méthode exacte , malgré l'efficacité et la précision de la solution mais il est déconseillé parcequ'il prend énormément de temps pour vérifier tous les chemins possibles, de l'autre coté les algorithmes de gluton donne des résultats proches de la solution optimale avec des temps d'exécution réduits (complexité polynomiale).

6. Conclusion :

Le problème du voyageur de commerce est toujours d'actualité dans la recherche en informatique, étant donné le nombre important de problèmes réels auxquels il correspond. Les problèmes dérivés et les extensions en sont très nombreux.