Algorithme du Point le Plus Proche pour le Problème du Voyageur de Commerce

Mohamed Elakef Zenagui

1 Introduction

Le problème du voyageur de commerce (*Travelling Salesman Problem*, TSP) consiste à trouver une tournée hamiltonienne de coût minimal qui passe exactement une fois par chaque sommet d'un graphe complet valué. Ce problème est connu pour être NP-difficile.

Dans ce travail, nous présentons une approche heuristique appelée **Point le Plus Proche (PPP)**, qui construit une solution approximative du TSP en ajoutant à chaque étape le point non encore inclus dans le cycle le plus proche du cycle existant.

2 Structures de données

CONSTANTE **n** = 10

TYPE **Arrete** = enregistrement
sommet : entier
suiv : ↑ Arrete
FIN

TYPE **GrapheM** = enregistrement
n : entier
M : Tableau[1..n][1..n] de **réel**
FIN

TYPE **cycle** = ↑ **Arrete**

3 Algorithme PPP

L'algorithme PPP construit un cycle hamiltonien en partant d'un point initial et en ajoutant successivement le point non inclus le plus proche du cycle courant, jusqu'à ce que tous les points soient inclus.

```
Input: G : GrapheM, s : Entier (point de départ)
Output: c : cycle (cycle hamiltonien)
Var u, v, min_dist : Entier
    marked : Tableau [1..n] de Booléen
c ← ∅
ajouter(c, s)
for i ← 1 to G.n do
| marked[i] ← Faux
end
marked[s] ← Vrai
while |c| ≠ G.n do
| min_dist ← inf
| for i ← 1 to G.n do
| | if not marked[i] then
| | | p ← c
| | | while p ≠ NIL do
| | | | if M[i, p ↑ sommet] < min_dist then
| | | | | min_dist ← M[i, p ↑ sommet]
| | | | | u ← i
| | | | | v ← p ↑ sommet
| | | | end
| | | | p ← p ↑ suiv
| | | end
| | end
| end
| ajouter(c, v, u) // Ajouter u dans le cycle c après v
| marked[u] ← Vrai
end
ajouter(c, s) // Fermer le cycle en revenant au point de départ
return c
```

Algorithm 1: Algorithme PPP

4 Amélioration de la stratégie du Point le Plus Proche

Une amélioration possible du coût du cycle obtenu par la procédure PPP consiste à **décroiser les arêtes qui se croisent**. Soit (i, j) un couple d'entiers dans l'intervalle $[1, n]$ tel que $j \geq i+2$, et soit le cycle :

$$c = (PL[1], \ldots, PL[i], PL[i+1], \ldots, PL[j], PL[j+1], \ldots, PL[n]).$$

Si le décroisement des arêtes $(PL[i], PL[i+1])$ et $(PL[j], PL[j+1])$ réduit la longueur totale du cycle, on transforme c en :

$$\bar{c} = (PL[1], \ldots, PL[i], PL[j], \ldots, PL[i+1], PL[j+1], \ldots, PL[n]).$$

```
Input: c : cycle obtenu par PPP
Output: c : cycle amélioré
Var amelioration : Booléen
    a, b, d, e : Entier
    p, q, r, s : ↑ Arrete
amelioration ← Vrai
while amelioration do
    amelioration ← Faux
    p ← c
    q ← p↑ suiv
    while q↑ suiv ≠ NIL do
        r ← q↑ suiv
        s ← r↑ suiv
        while s ≠ NIL do
            a ← p↑ sommet, b ← q↑ sommet
            d ← r↑ sommet, e ← s↑ sommet
            if G.M[a, d] + G.M[b, e] < G.M[a, b] + G.M[d, e] then
                // Décroisement avantageux
                // Inverser le segment de c compris entre les cellules q et r
                inverser(c, q, r)
                amelioration ← Vrai
            end
            r ← s
            s ← s↑ suiv
        end
        p ← q
        q ← q↑ suiv
    end
end
return c
```

Algorithm 2: Algorithme OptPPP

Cet algorithme répète les opérations de décroisement tant qu'il existe des couples d'arêtes croisées dont le remplacement réduit la longueur du cycle. Il permet ainsi d'améliorer efficacement la solution initiale fournie par PPP.

5 Conclusion

L'algorithme PPP est une heuristique simple et efficace pour générer une solution approximative au TSP. Bien qu'il ne garantisse pas d'obtenir le cycle de coût minimal, il fournit rapidement une solution valide et peut être utilisé comme point de départ pour d'autres méthodes d'optimisation.