Stratégie par l'arbre couvrant de poids minimum

Mohamed Elakef Zenagui

1 Introduction

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

Dans ce travail, nous présentons un algorithme exact basé sur le principe du *Branch and Bound*, utilisant une borne inférieure obtenue par l'heuristique de la demi-somme, noté **HDS**.

2 Structures de données

CONSTANTE **n** = 10

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

TYPE **GrapheD** = enregistrement
n : entier
L : Tableau[1..n] de ↑ *Arrete*
clé : Tableau[1..n] de **réel**
π : Tableau[1..n] d'**entier**
NoeudTas : Tableau[1..n] d'**entier**
FIN

TYPE **Tas** = enregistrement
dern : entier
Tab : Tableau[1..n] d'**entier**
FIN

TYPE **Cellule** = enregistrement
sommet : entier // sommet courant
suiv : ↑ Cellule // cellule suivante
FIN

TYPE **Acellule** = ↑ Cellule

TYPE **GrapheTL** = enregistrement
n : entier // nombre de sommets

L : Tableau[1..n] de Acellule // listes de successeurs
FIN

TYPE **COULEUR** = (BLANC, GRIS, NOIR)

3 Parcours en profondeur

Input: GrapheTL G, Entier $origine$
Output: Tableau $couleurs[1..n]$, $pi[1..n]$, $P[1..n]$, $S[1..n]$
Output: Tableau $P^*[1..n]$, $S^*[1..n]$
Output: Entiers i_p, i_s
Var v : Entier
p : Acellule

```
couleurs[origine] ← GRIS
i_p ← i_p + 1
P[origine] ← i_p
P*[i_p] ← origine
p ← G.L[origine]
while p ≠ NIL do
    v ← p ↑ sommet
    if couleurs[v] = BLANC then
        pi[v] ← origine
        Visiter_Profondeur(G, v, couleurs, pi, P, S, P*, S*, i_p, i_s)
    end
    p ← p ↑ suiv
end
couleurs[origine] ← NOIR
i_s ← i_s + 1
S[origine] ← i_s
S*[i_s] ← origine
```

Algorithm 1: Visite en profondeur

4 Algorithme principal (backtracking)

Input: GrapheTL G
Output: Tableaux pi, P, S, P^*, S^*
Vari, i_p, i_s : $Entier$
couleurs : Tableau[1..G.n] de COULEUR

```
for i ← 1 to G.n do
    couleurs[i] ← BLANC
    pi[i] ← −1
    P[i] ← 0
    S[i] ← 0
    P*[i] ← 0
    S*[i] ← 0
end
i_p ← 0
i_s ← 0
for i ← 1 to G.n do
    if couleurs[i] = BLANC then
        Visiter_Profondeur(G, i, couleurs, pi, P, S, P*, S*, i_p, i_s)
    end
end
```

Algorithm 2: Parcours en profondeur avec backtracking

5 Version optimal du Prim

L'heuristique de la demi-somme fournit une borne inférieure du coût restant à parcourir. Pour chaque sommet on considère la demi-somme des deux plus petites arêtes incidentes admissibles.

Input: G : GrapheD, r : Entier
Output: π : Tableau[1..n] d'Entier
Var t, i : Entier
  p : $\uparrow$ *Arrete*

$T.dern \leftarrow 0$
for $i \leftarrow 1$ **to** $G.n$ **do**
  $G.cle[i] \leftarrow \text{inf}$
  $G.\pi[i] \leftarrow -1$
  $G.NoeudTas[i] \leftarrow i$
  $T.Tab[i] \leftarrow i$
end
$G.cle[r] \leftarrow 0$
while $T.dern \geq 1$ **do**
  if $p = c$ ***ou*** $q \uparrow suiv = NIL$ **then**
    $t \leftarrow T.Tab[1]$
    Echange$(1, T.dern, G, T)$
    $T.dern \leftarrow T.dern - 1$
    Verslebas$(1, G, T)$
    $p \leftarrow G.L[t]$
    while $p \neq NIL$ **do**
      if $G.NoeudTas[t] \leq T.dern$ ***et*** $p \uparrow poids < G.cle[p \uparrow sommet]$ **then**
        $G.cle[p \uparrow sommet] \leftarrow p \uparrow poids$
        $G.\pi[p \uparrow sommet] \leftarrow t$
        Verslehaut$(G.NoeudTas[p \uparrow sommet], G, T)$
      end
      $p \leftarrow p \uparrow suiv$
    end
  end
end

Algorithm 3: Prim Efficace

6 Algorithme HDS

L'algorithme HDS explore l'espace des solutions partielles en privilégiant les nœuds possédant la plus petite borne inférieure.

```
Input: GrapheM G
Output: ↑ Arrete best_solution, Réel best_cost
Var u, v, n : Entier
    borne, h : Réel
    T : Tas
    c, new_c : ↑ Arrete
n ← |M|
best_cost ← +∞
best_solution ← NIL
T ← ∅
c ← ∅
ajouter(c, 1)
cost ← 0
borne ← Heuristique_Demi_Somme(M, c)
Entasser(T, ⟨borne, cost, c⟩)
while T ≠ NIL do
    Détasser(T, ⟨borne, cost, c⟩)
    if borne < best_cost then
        if |c| = n then
            total_cost ← cost + M[dernier(c)][1]
            if total_cost < best_cost then
                best_cost ← total_cost
                best_solution ← c
            end
        end
        else
            u ← dernier(c)
            for v ← 1 to n do
                if not v ∈ sommets(c) then
                    new_cost ← cost + M[u][v]
                    new_c ← c
                    ajouter(new_c, v)
                    h ← Heuristique_Demi_Somme(M, new_c)
                    if h < best_cost then
                        Entasser(T, ⟨h, new_cost, new_c⟩)
                    end
                end
            end
        end
    end
end
return (best_solution, best_cost)
```

Algorithm 4: Algorithme HDS

7 Conclusion

L'algorithme HDS est un algorithme exact pour la résolution du TSP. L'utilisation de l'heuristique de la demi-somme permet d'obtenir une borne inférieure admissible, réduisant considérablement l'espace de recherche par élagage.