
Fonction 1 Algorithme de séparation et évaluation progressive (*branch and bound*)

```
1: function BRANCH_AND_BOUND(graphe)  $\triangleright \mathcal{O}(n^3 \cdot \sum_{i=1}^n i!)$ 
2:   pile  $\leftarrow \emptyset$ 
3:   nombre_obstructions_min  $\leftarrow \infty$ 

4:   voisins_triés  $\leftarrow \emptyset$ 
5:   for all noeud  $\in$  graphe do  $\triangleright n$  itérations pour  $n$  noeuds  $\Rightarrow \Theta(n^2 \log n)$ 
6:     voisins_triés  $\leftarrow \{\text{voisins}(\text{noeud}) \text{ triés par hauteur croissante}\}$   $\triangleright \Theta(n \log n)$ 
7:   end for

8:   while pile  $\neq \emptyset$  do
9:      $\triangleright \text{nombre d'itérations pour } n \text{ noeuds} \leq \sum_{i=1}^n i! \Rightarrow \mathcal{O}(n^3 \cdot \sum_{i=1}^n i!)$ 
     chemin  $\leftarrow \text{dépiler}(\text{pile})$   $\triangleright \Theta(1)$ 

10:    if longueur(chemin)  $\geq$  nombre_obstructions_min then
11:      nombre_obstructions  $\leftarrow \text{COMPTER\_OBSTRUCTIONS}(\text{chemin})$   $\triangleright \Theta(n)$ 
12:      if nombre_obstructions  $\geq$  nombre_obstructions_min then  $\triangleright \Theta(1)$ 
13:        continue
14:      end if

15:      if longueur(chemin) = |graphe| then  $\triangleright \Theta(1)$ 
16:        afficher solution  $\triangleright \Theta(n)$ 
17:        nombre_obstructions_min  $\leftarrow$  nombre_obstructions
18:        continue
19:      end if
20:    end if

21:    nouveaux_chemins  $\leftarrow \text{ÉTENDRE\_CHEMIN}(\text{chemin})$   $\triangleright \mathcal{O}(n^3)$ 
22:    empiler(pile, nouveaux_chemins)  $\triangleright |\text{nouveaux\_chemins}| < n \Rightarrow \mathcal{O}(n)$ 
23:  end while
24: end function
```

(1)

Fonction 2 Extension du chemin partiel en lui ajoutant un nouveau noeud

```

1: function ÉTENDRE_CHEMIN(chemin : [noeuds], voisins_triés : map)    ▷  $\mathcal{O}(n^3)$ 
2:   nouveaux_chemins  $\leftarrow \emptyset$ 

3:   voisins  $\leftarrow$  voisins_triés[dernier sommet de chemin]          ▷  $\Theta(1)$ 
4:   for all voisin  $\in$  voisins \ chemin do                            ▷  $|voisins| < n \Rightarrow \mathcal{O}(n^3)$ 
5:     a_un_voisin_condamné  $\leftarrow$  false

6:     if  $\exists \geq 2$  noeuds  $\notin$  chemin then
7:       for all voisin_degré_2  $\in$  voisins(voisin) \ chemin do
8:         voisins_deg3_non_visités  $\leftarrow 0$ 
9:         for all voisin_degré_3  $\in$  voisins(voisin_degré_2) \ chemin do
10:          voisins_deg3_non_visités  $\leftarrow$  voisins_deg3_non_visités + 1
11:          if voisins_deg3_non_visités > 1 then                            ▷  $\Theta(1)$ 
12:            break
13:          end if
14:        end for

15:        if voisins_deg3_non_visités  $\leq 1$  then                            ▷  $\Theta(1)$ 
16:          a_un_voisin_condamné  $\leftarrow$  true
17:          break
18:        end if

19:      end for
20:    end if

21:    if  $\neg a\_un\_voisin\_condamné$  then
22:      nouveaux_chemins  $\leftarrow$  nouveaux_chemins  $\cup \{(chemin \cup voisin)\}$ 
23:      ▷  $|chemin| + 1 < n \Rightarrow \mathcal{O}(n)$ 
24:    end if
25:  end for

26:  return nouveaux_chemins
27: end function

```

Fonction 3 Comptage du nombre d'obstructions dans une solution

```
1: function COMPTE_OBSTRACTIONS(chemin : [noeuds]) ▷  $\Theta(n)$ 
2:   nombre_obstructions ← 0
3:   hauteur_max ← 0
4:   for all noeud ∈ chemin do ▷  $n$  itérations pour  $n$  noeuds ⇒  $\Theta(n)$ 
5:     if hauteur(noeud) < hauteur_max then ▷ op. baromètre :  $\Theta(1)$ 
6:       nombre_obstructions ← nombre_obstructions + 1
7:     else
8:       hauteur_max ← hauteur(noeud)
9:     end if
10:  end for
11:  return nombre_obstructions
12: end function
```
