Chapitre 7 : Parcours de graphe

INF4705 - Analyse et conception d'algorithmes

Gilles Pesant   Simon Brockbank

École Polytechnique Montréal
gilles.pesant@polymtl.ca, simon.brockbank@polymtl.ca

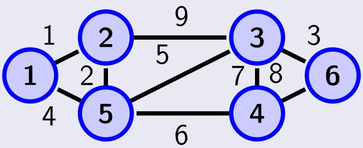
Hiver 2017

Plan

Motivation

De nombreux problèmes sont représentés à l'aide d'un graphe.
Pour les résoudre, on a souvent besoin de parcourir ce graphe.

Graphe explicite



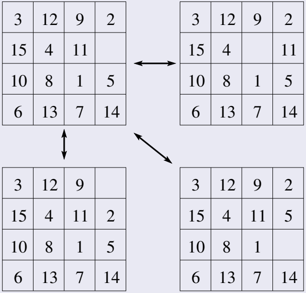
Graphe implicite

Pour un problème combinatoire, représente l'espace des configurations possibles.
Trop gros pour le garder en mémoire ; on n'en manipule qu'une petite partie à la fois.

Graphe implicite ; configurations totales

Ex : Jeu de Taquin

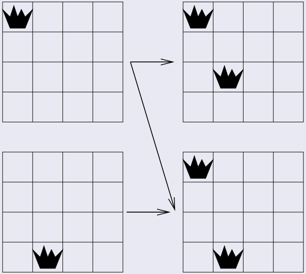
Déplacer les tuiles en les glissant afin de les placer en ordre numérique.



Graphe implicite ; configurations partielles

Ex : *n*-reines

Placer n reines sur un échiquier de taille $n \times n$ de telle façon qu'elles ne se menacent pas (ligne, colonne et diagonale différentes).



Plan

Fouille en profondeur

Patron de conception (formulation itérative)

procédure fep(v : sommet) {avec *pile* P}
  P $\leftarrow \emptyset$;
  visité[v] $\leftarrow$ vrai ;
  empiler(v,P) ;
  tant que P $\neq \emptyset$ **faire**
    tant que $\exists w$ adjacent à **haut**(P) t.q. non visité[w] **faire**
      visité[w] $\leftarrow$ vrai ;
      empiler(w,P) ;
    dépiler(P) ;

Fouille en largeur

Patron de conception (formulation itérative)

procédure fel(v : sommet) {avec *file* F}

    F $\leftarrow \emptyset$;
    visité[v] $\leftarrow$ vrai ;
    mettre(v,F) ;
    tant que F $\neq \emptyset$ **faire**
        tant que $\exists w$ adjacent à **premier**(F) t.q. non visité[w]
        faire
            visité[w] $\leftarrow$ vrai ;
            mettre(w,F) ;
        sortir(F) ;

Et si le graphe n'est pas connexe ?

procédure parcours($G = (N, A)$: graphe, fouille : fep ou fel)
  $\forall v \in N$ **faire**
    visité[v] $\leftarrow$ faux ;
  $\forall v \in N$ **faire**
    si non visité[v] **alors** fouille(v) ;

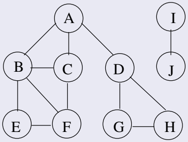
Soit un graphe avec n sommets et a arêtes (arcs).
Temps de calcul pour le parcourir $\in \Theta(\max(n, a))$:
un sommet est mis une seule fois sur la pile/file ;
une arête est considérée deux fois (et un arc, une seule fois).

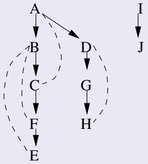
Classification des arêtes selon leur rôle pendant la fouille

graphe non-orienté

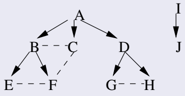
- arêtes d'arbre utilisées par la fouille et formant une forêt
- [fep] toutes les autres, arêtes arrières :
  joignent un sommet à un de ses ancêtres
- [fel] toutes les autres, arêtes latérales :
  joignent deux sommets sans lien de descendance

Exemple de parcours





fep: A,B,C,F,E,D,G,H,I,J



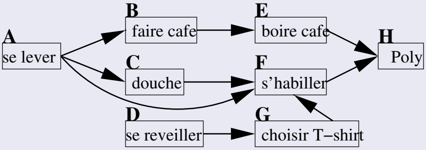
fel: A,B,C,D,E,F,G,H,I,J

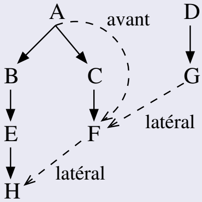
Classification des arêtes selon leur rôle pendant la fouille

graphe orienté

- [fep] arcs d'arbre, arrières, latéraux et avants (d'un sommet vers un de ses descendants)
- [fel] arcs d'arbre, arrières et latéraux

Exemple de parcours





fep : A,B,E,H,C,F,D,G

Quelques problèmes sur un graphe non-orienté

Est-ce que G contient un cycle ?

Si $a \geq n$ alors on a nécessairement un cycle.
Sinon ($a < n$) :

- parcours(G,fep) : $\exists$ cycle ssi $\exists$ arête arrière.
- parcours(G,fel) : $\exists$ cycle ssi $\exists$ arête latérale.

On répond donc en $\Theta(n)$.

G est-il connexe ?

ssi parcours(G,fep) ou parcours(G,fel) génère un seul arbre

Sinon, quelles sont ses composantes connexes ?

chaque arbre obtenu de parcours(G,fep) ou parcours(G,fel) correspond à une composante connexe.

Quelques problèmes sur un graphe non-orienté

Couplage maximum dans un graphe biparti

Un *couplage* dans un graphe est un sous-ensemble de ses arêtes qui ne partagent aucun sommet. Un couplage est *maximum* s'il n'en existe pas d'autre avec plus d'arêtes.

Chemin augmentant

Étant donné un certain couplage C dans le graphe, un *chemin augmentant* relie deux sommets non couplés dans C et est composé en alternance d'arêtes faisant partie et ne faisant pas partie de C. Un tel chemin est nécessairement de longueur impaire.

Théorème

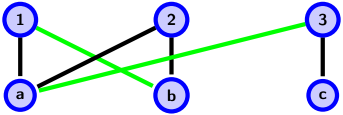
Un couplage est maximum ssi il n'existe pas de chemin augmentant.

Quelques problèmes sur un graphe non-orienté

Algorithme de couplage maximum

fonction couplageMax($G = (X, Y, A)$: graphe biparti) : couplage
$C \leftarrow \emptyset$;
tant que $\exists$ chemin augmentant, disons γ, **faire**
$C \leftarrow C \oplus \gamma$;
retourner C

On cherche un chemin augmentant en appliquant une fel à partir des sommets non couplés jusqu'à ce qu'on atteigne un autre sommet non couplé.



Quelques problèmes sur un graphe non-orienté

Algorithme de couplage maximum

fonction couplageMax($G = (X, Y, A)$: graphe biparti) : couplage

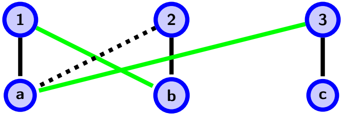
$C \leftarrow \emptyset$;

tant que $\exists$ chemin augmentant, disons γ, **faire**

$C \leftarrow C \oplus \gamma$;

retourner C

On cherche un chemin augmentant en appliquant une fel à partir des sommets non couplés jusqu'à ce qu'on atteigne un autre sommet non couplé.

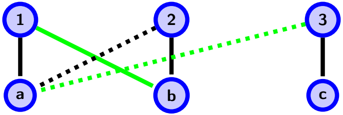


Quelques problèmes sur un graphe non-orienté

Algorithme de couplage maximum

fonction couplageMax($G = (X, Y, A)$: graphe biparti) : couplage
$C \leftarrow \emptyset$;
tant que $\exists$ chemin augmentant, disons γ, **faire**
$C \leftarrow C \oplus \gamma$;
retourner C

On cherche un chemin augmentant en appliquant une fel à partir des sommets non couplés jusqu'à ce qu'on atteigne un autre sommet non couplé.

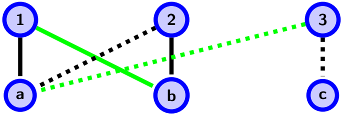


Quelques problèmes sur un graphe non-orienté

Algorithme de couplage maximum

fonction couplageMax($G = (X, Y, A)$: graphe biparti) : couplage
  $C \leftarrow \emptyset$;
  tant que $\exists$ chemin augmentant, disons γ, **faire**
    $C \leftarrow C \oplus \gamma$;
  retourner C

On cherche un chemin augmentant en appliquant une fel à partir des sommets non couplés jusqu'à ce qu'on atteigne un autre sommet non couplé.

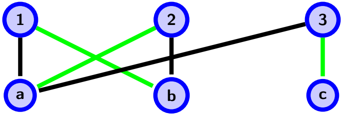


Quelques problèmes sur un graphe non-orienté

Algorithme de couplage maximum

fonction couplageMax($G = (X, Y, A)$: graphe biparti) : couplage
  $C \leftarrow \emptyset$;
  tant que $\exists$ chemin augmentant, disons γ, **faire**
    $C \leftarrow C \oplus \gamma$;
  retourner C

On cherche un chemin augmentant en appliquant une fel à partir des sommets non couplés jusqu'à ce qu'on atteigne un autre sommet non couplé.



Quelques problèmes sur un graphe orienté

Est-ce que G contient un cycle ?

parcours(G,fep ou fel) : $\exists$ cycle ssi $\exists$ arête arrière.

G est-il fortement connexe ? Sinon quelles sont ses composantes ?

1. parcours(G,fep*)
2. parcours*($G^{\top}$,fep)

Chaque arbre obtenu correspond à une composante fortement connexe.

Fouille en profondeur modifiée

procédure fep*(v : sommet)
  P $\leftarrow \emptyset$;
  Q $\leftarrow \emptyset$;
  visité[v] $\leftarrow$ vrai ;
  empiler(v,P) ;
  tant que P $\neq \emptyset$ **faire**
    tant que $\exists w$ adjacent à **haut**(P) t.q. non visité[w] **faire**
      visité[w] $\leftarrow$ vrai ;
      empiler(w,P) ;
    empiler(**haut**(P),Q) ;
    dépiler(P) ;

parcours modifié

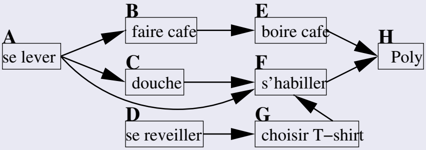
procédure parcours*($G = (N, A)$: graphe, fep)
    $\forall v \in N$ **faire**
        visité[v] $\leftarrow$ faux ;
    tant que Q $\neq \emptyset$ **faire**
        $v \leftarrow$ **haut**(Q) ;
        dépiler(Q) ;
        si non visité[v] **alors** fouille(v) ;

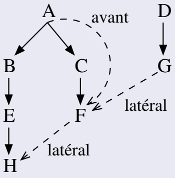
Quelques problèmes sur un graphe orienté

Tri topologique d'un graphe sans cycle

Ordonner ses sommets tel que $\forall$ arc (u, v) u précède v.

parcours(G,fep*)
tant que Q $\neq \emptyset$ **faire**
  écrire **haut**(Q) ;
  dépiler(Q) ;





D,G,A,C,F,B,E,H

Parcours pré-ordre

On visite le sommet lui-même,
puis récursivement les sous-arbres de ses enfants.

procédure pré-ordre(u : sommet)
  {Soient $v_1, \ldots, v_k$ les enfants de u}
  visiter u ;
  pour $i \leftarrow 1$ **à** k **faire**
    pré-ordre(v_i) ;

Parcours post-ordre

On visite d'abord récursivement les sous-arbres de ses enfants, puis le sommet lui-même.

procédure post-ordre(u : sommet)
    {Soient $v_1, \ldots, v_k$ les enfants de u}
    pour $i \leftarrow 1$ **à** k **faire**
        post-ordre(v_i) ;
    visiter u ;

Parcours en-ordre

On visite récursivement le sous-arbre de son premier enfant,
ensuite le sommet lui-même,
puis récursivement les sous-arbres de ses autres enfants.

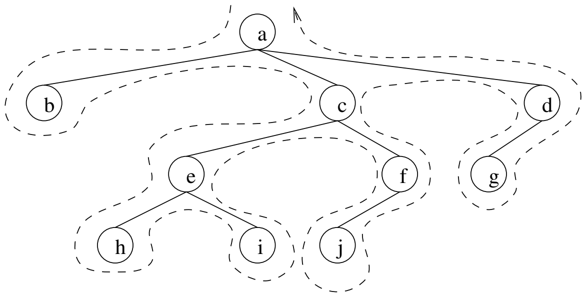
procédure en-ordre(u : sommet)
    {Soient $v_1, \ldots, v_k$ les enfants de u}
    si $k = 0$ {u est une feuille}
        alors visiter u ;
        sinon
            en-ordre(v_1) ;
            visiter u ;
            pour $i \leftarrow 2$ **à** k **faire**
                en-ordre(v_i) ;

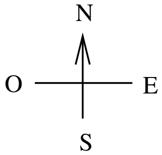
Parcours par niveau

Si on applique une fouille en largeur à un arbre, on obtient un parcours par niveau.

Détermination facile des parcours d'arbre

pré-ordre : fenêtre ouest ; a,b,c,e,h,i,f,j,d,g
post-ordre : fenêtre est ; b,h,i,e,j,f,c,g,d,a
en-ordre : fenêtre sud ; b,a,h,e,i,c,j,f,g,d

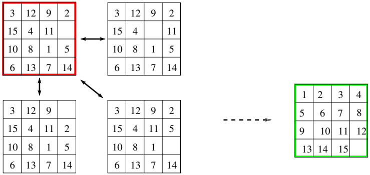




Plan

Configurations totales

Quand le graphe implicite est bâti sur des configurations totales avec une configuration initiale et une configuration cible (ex. Jeu de Taquin, cube Rubik), on cherche généralement la solution nécessitant le moins d'étapes (le plus court chemin).



Fouille en largeur

Par définition, lorsque la configuration cible est atteinte il n'y a pas de plus court chemin à celle-ci.

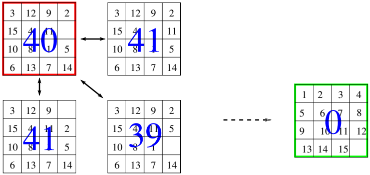
Configurations totales

Meilleur d'abord ("best-first")

fel + expansion du graphe selon une évaluation heuristique de la distance à la configuration cible

Ne garantit pas que le (premier) chemin trouvé est le plus court

Ex : Pour le Jeu de Taquin, la "somme des distances manhattan de chacune des tuiles à sa position finale"



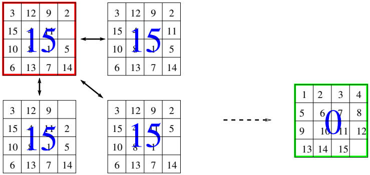
Configurations totales

A*

meilleur d'abord mais selon une heuristique *admissible* (donne une borne inférieure de la distance à la configuration cible)

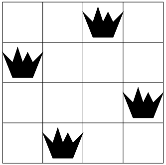
Garantit que le chemin trouvé est le plus court

Ex : Pour le Jeu de Taquin, le "nombre de tuiles qui ne sont pas à leur place"



Configurations partielles

On ne connaît pas la configuration cible ; c'est elle qu'on cherche.



On pourrait *générer-puis-tester* les configurations totales
$\Rightarrow$ très inefficace

Fouille à retour arrière ("backtracking")

fep sur graphe implicite de configurations partielles

Plus efficace car élimine implicitement plusieurs configurations totales à la fois.

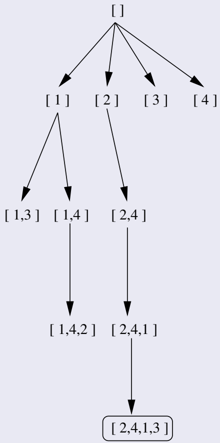
Fouille à retour arrière ("backtracking")

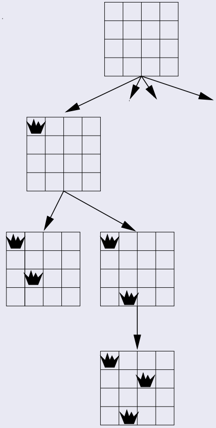
Ex : *n*-reines

- configuration partielle $[\ell_1, \ldots, \ell_k]$, $0 \leq k \leq n$ indique la ligne sur laquelle est placée chacune des reines des colonnes 1 à k
- configuration partielle $[\ell_1, \ldots, \ell_k]$ légale si aucune des k reines ne se menace
- solution : configuration légale $[\ell_1, \ldots, \ell_n]$ avec $k = n$ (donc totale)
- graphe orienté sur les configurations partielles légales avec arcs $([\ell_1, \ldots, \ell_k],\ [\ell_1, \ldots, \ell_k, \ell_{k+1}])$

Fouille à retour arrière ("backtracking")

Ex : 4-reines





Séparation et évaluation progressive ("branch-and-bound")

Dans le contexe d'un problème d'optimisation (spdg maximisation) :

Séparation et évaluation progressive ("branch-and-bound")

fouille à retour arrière + à chaque configuration partielle, calcul d'une borne (supérieure) sur la valeur de toutes les solutions potentielles dans le sous-arbre

Nous nous servons de la borne pour éliminer des sous-arbres sous-optimaux dans notre arbre de recherche.

(Nous pouvons aussi nous en servir pour guider l'exploration de cet arbre, comme avec meilleur d'abord.)

Séparation et évaluation progressive ("branch-and-bound")

Ex : Sac à dos

$$\max \Sigma_{i=1}^{n} x_i v_i, \quad x_i \in \{0,1\}$$

$$\text{tel que } \Sigma_{i=1}^{n} x_i w_i \leq W$$

graphe implicite sur configurations partielles
$[x_1, \ldots, x_k]$, $0 \leq k \leq n$, légales (ne dépassent pas la capacité W)

Exemple : Sac à dos

Borne supérieure A

À la configuration partielle $x_1, \ldots, x_k$:

$$\Sigma_{i=1}^{k} x_i v_i + (W - \Sigma_{i=1}^{k} x_i w_i) \cdot v_j / w_j, \quad \text{où } j = \text{argmax}_{k+1 \leq i \leq n} v_i / w_i$$

Borne supérieure B

À la configuration partielle $x_1, \ldots, x_k$:

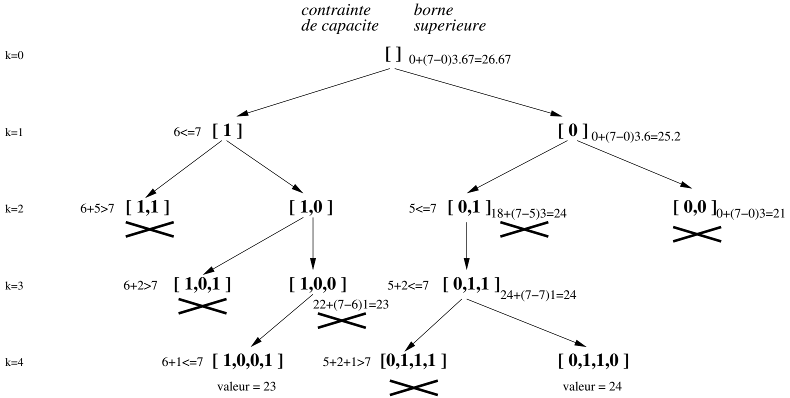
$$\Sigma_{i=1}^{k} x_i v_i + \text{glouton}(0 \leq x_i \leq 1, k+1 \leq i \leq n, W - \Sigma_{i=1}^{k} x_i w_i)$$

borne B $\leq$ borne A

Exemple : Sac à dos

$W = 7$; borne A

objet	valeur	poids	v_i / w_i
1	22	6	3.67
2	18	5	3.60
3	6	2	3.00
4	1	1	1.00



Séparation et évaluation progressive ("branch-and-bound")

Améliorations possibles

- meilleures bornes
- *heuristiques de branchement* :
  - choix dynamique du prochain élément pour l'expansion d'une configuration partielle
  - ordre d'exploration des branches
- *anticipation/inférence* : on peut réduire la taille du sous-arbre à un sommet en anticipant les impasses ("forward-checking", "look-ahead", techniques de cohérence)
- *apprentissage* des impasses