Recherches dans les graphes

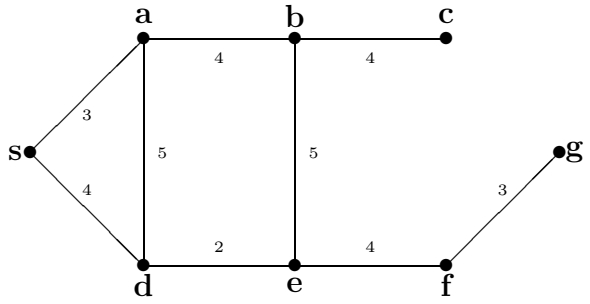
P. De Loor, P.A. Favier, J. Tisseau

November 18, 2002

1 Exemple de réseau routier

Dans les 3 sections suivantes (2 page 2 , 3 page 4 , 4 page 5), nous considèrerons le graphe du réseau routier[1] suivant :

```
route(s,a,3).
route(s,d,4).
route(b,a,4).
route(b,c,4).
route(b,e,5).
route(d,a,5).
route(e,d,2).
route(f,e,4).
route(f,g,3).
```



réseau routier

```
estim(a,g,10.4).  estim(b,g,6.7).  estim(c,g,4.0).  estim(d,g, 8.9).
estim(e,g, 6.9).  estim(f,g,3.0).  estim(g,g,0.0).  estim(s,g,12.5).
```

route(?Ville1,?Ville2,?N)

vrai si et seulement si il existe une route directe de **N** *km* entre **Ville1** et **Ville2**.

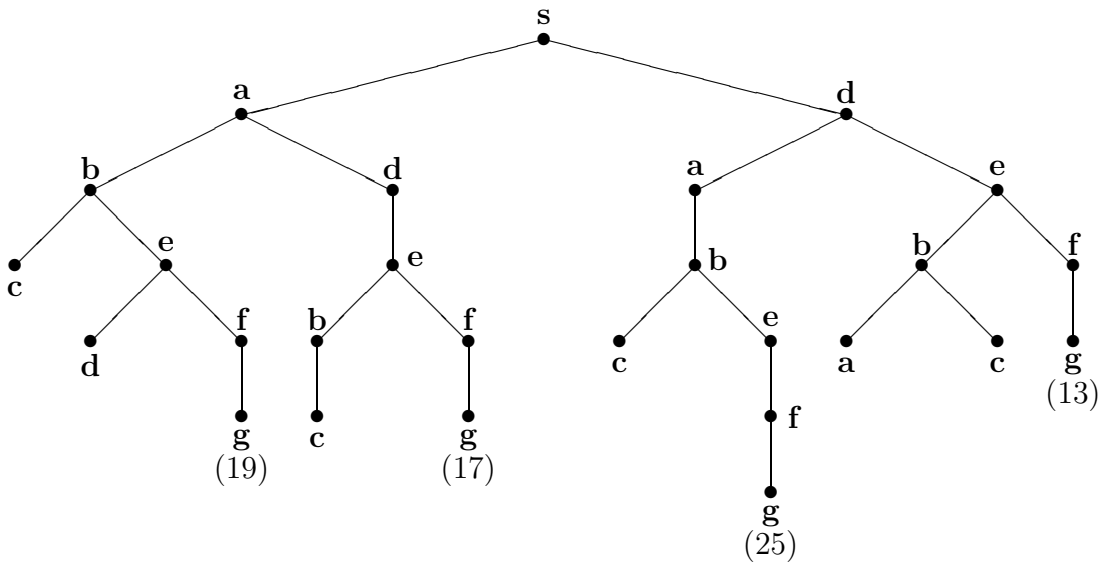
estim(?Ville1,?Ville2,?N)

vrai si et seulement si la distance à vol d'oiseau entre **Ville1** et **Ville2** est de **N** *km*.

Nous chercherons par la suite un chemin pour aller de **s** en **g**; c'est pourquoi nous ne donnons ici que les distances à vol d'oiseau entre une ville quelconque du réseau et la ville **g** .

Définir le prédicat **successeur(?Ville1,?Ville2,?N)** vrai si et seulement si **Ville1** et **Ville2** sont reliées par une route directe de **N** *km* .
L'arbre de recherche associé à ce réseau, lorsqu'on veut aller de **s** à **g**, est représenté ci–dessous.

[1] Cet exemple est utilisé dans [**?**] pour illustrer les différents algorithmes de recherche dans les graphes.



2 Représentation d'un état

Un état du graphe à un instant donné sera représenté par la liste PROLOG de ses branches. Une branche sera représenté par un terme du type

Cout-Passe/Noeud

où **Cout** est le coût (réel ou estimé) du **Noeud** compte–tenu des nœuds par lesquels on est déjà passé (**Passe**) pour y arriver.
Passe est du type **X1/X2/X3/ ... /Xn**, où les X_i sont des nœuds.
Exemples :

- **0-s** représente une branche possédant un seul noeud (le noeud de départ par rapport à la figure).
- **12-s/a/b/e** représente la branche se terminant par **e** en passant par les nœuds **s**, **a**, et **b**, dans cet ordre, pour un coût réel de 12 km .
- **18-s/d/a/b/e** représente également une branche se terminant par le nœud **e**; mais cette fois, il a été atteint en passant par les nœuds **s**, **d**, **a**, et **b**, dans cet ordre, pour un coût réel de 18 km .

Définir les prédicats suivants :

1. **sommet(+NoeudDepart,?EtatDepart)**

   vrai si et seulement si **EtatDepart** représente l'état du graphe (liste composée d'une seule branche) pour le nœud initial **NoeudDepart**.

2. **informationsBranche(+Branche,?Chemin,?Cout,?Noeud)**

   vrai si et seulement si **Chemin** est le chemin parcouru d'extrémité **Noeud** et de coût **Cout** pour la branche**Branche** considéré.

3. **informationsEtatGraphe(+Etat,?Premier,?Reste)**

   vrai si et seulement si **Premier** et **Reste** sont respectivement la premire branche et le reste des branches à considérer dans l'**Etat** spécifié.

Compte–tenu de ces définitions, la recherche dans un graphe peut s'écrire sous la forme générale suivante (**chemin/4** et **chemin1/4**) :

```
chemin(NoeudDepart,NoeudArrivee,Chemin,Cout) :-
  sommet(NoeudDepart,EtatDepart),
  chemin1(EtatDepart,Chemin,Cout,NoeudArrivee).

chemin1(Etat,Chemin,Cout,Arrivee) :-
  informationsEtatGraphe(Etat,PremiereBranche,ResteBranches),
  informationsBranche(PremiereBranche,Chemin,Cout,Arrivee).

chemin1(Etat,Chemin,Cout,Arrivee) :-
  informationsEtatGraphe(Etat,PremiereBranche,ResteBranches),
  candidatsSuivants(PremiereBranche,BranchesSuivantes,Arrivee),
  transformationEtatGraphe(BranchesSuivantes,ResteBranches,EtatSuivant,
                      Arrivee),
  chemin1(EtatSuivant,Chemin,Cout,Arrivee).
```

candidatsSuivants(+Branche,?BranchesSuivantes,+NoeudArrivee)

vrai si et seulement si **BranchesSuivants** est la liste des branches commencant par **Branche** et possédant un noeud de plus. Ce noeud doit bien sûr être acceéssible à partir de branche. Les boucles sont à exclure (le nouveau noeud ne doit pas déjà être dans **Branche**.

NoeudArrivee sera nécessaire si l'on a besoin de connaître la distance au but d'un nœud (comme dans la recherche en profondeur orientée).

Exemple :

```
?- candidatsSuivants(12-s/a/b, BS, _).

BS = [16-s/a/b/c, 17-s/a/b/e]

Yes
```

transformationEtatGraphe(+Etat,+NouvelleBranche,?EtatSuivant,+Arrivee)

vrai si et seulement si **EtatSuivant** est le nouvel état du graphe avec les nouvelles branches **NouvelleBranche**). Le nœud **Arrivee** sera nécessaire si l'on a besoin de connaître la distance au but d'un nœud (comme dans la recherche du meilleur premier ou en faisceau).

Ces deux prédicats dépendent du problème considéré (réseau routier, jeu du taquin, ...) et du type de recherche envisagé (recherche en profondeur, recherche en largeur, ...).

3 Recherche d'un chemin quelconque

1. Dans l'exemple du réseau routier, définir le prédicat **candidat(+Branche, ?CandidatSuivant)** vrai si et seulement si **CandidatSuivant** est un candidat directement accessible depuis **Branche = Cout-Passe/Noeud**), et par lequel on n'est pas déjà passé pour arriver au **Noeud** (pas de cycle).

2. Définir les prédicats **candidatsSuivants/3** et **transformationEtatGraphe/4** pour les 5 types de recherche suivants :

   (a) **Recherche en profondeur** (pile de candidats).

   (b) **Recherche en profondeur orientée :** recherche en profondeur dans laquelle les nœuds immédiatement suivants d'un nœud candidat sont classés par ordre croissant de leurs distances au but (distances à vol d'oiseau).

   (c) **Recherche en largeur** (file de candidats).

   (d) **Recherche du meilleur premier :** recherche en largeur dans laquelle la liste des nœuds candidats d'un état du graphe est classée par ordre croissant des distances au but (distances à vol d'oiseau).

   (e) **Recherche en faisceau :** recherche en largeur dans laquelle on ne garde à chaque niveau que les deux meilleurs nœuds candidats du point de vue de leurs distances au but (distances à vol d'oiseau).

3. Chacune des recherches précédentes conduit à plusieurs résultats. Déterminer le meilleur chemin parmi tous les chemins obtenus pour chaque type de recherche.

4. Appliquer ces différents types de recherche au problème du remplissage de deux récipients.

   On dispose de deux récipients vides ($V_1 = V_2 = 0l$), non gradués, de capacités respectives $C_1 = 8l$ et $C_2 = 5l$.

   Les seules opérations possibles consistent à :

   - remplir complètement un récipient dans la mer ($V_i \rightarrow C_i$);
   - vider complètement un récipient dans la mer ($V_i \rightarrow 0l$);
   - transvaser, sans perte, un récipient dans l'autre jusqu'à ce que le deuxième récipient soit plein, ou que le premier soit vide ($V_1 + V_2 = C^{te}$).

L'état final recherché est celui où le premier récipient est à moitié vide ($V_1 = 4l$), et le deuxième vide ($V_2 = 0l$).

(a) Un nœud du graphe associé sera représenté par un doublet (V_1, V_2), où V_1 et V_2 sont les volumes d'eau contenus respectivement dans les deux récipients.

Définir le prédicat **successeur(?Noeud1,?Noeud2,?N)** vrai si et seulement si **Noeud2** est un suivant immédiat de **Noeud1** par application d'une des trois opérations permises (**N** = 1 dans tous les cas).

(b) Utiliser les différents types de recherche pour trouver un chemin allant du nœud **(0,0)** au nœud **(4,0)**.

4 Recherche du meilleur chemin

1. Définir les prédicats **candidatsSuivants/3** et **transformationEtatGraphe/4** pour les 4 types de recherche suivants :

   (a) **Recherche au moindre coût :** recherche en largeur dans laquelle la liste des nœuds candidats d'un état du graphe est classée par ordre croissant des coûts réels (distances déjà parcourues).

   (b) **Recherche au moindre coût avec élimination des doublons :** recherche au moindre coût dans laquelle on ne garde à chaque étape que le meilleur représentant (du point de vue des coûts réels) d'un nœud donné.

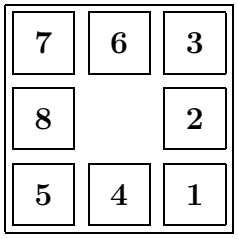
   (c) **Recherche heuristique :** recherche au moindre coût dans laquelle, pour chaque nœud, le coût est le coût total estimé (distance déjà parcourue + distance au but).

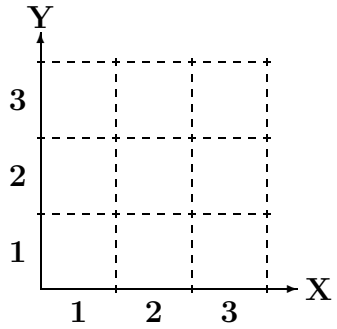
   (d) **Recherche heuristique par la procédure A* :** la procédure A^* combine la recherche heuristique précédente (classement des nœuds candidats par ordre croissant du coût total estimé) et l'élimination des doublons.

2. Appliquer ces différents types de recherche au problème du jeu du taquin à 9 cases (8 pavés numérotés de **1** à **8**).

   Un nœud du graphe sera représenté par une liste de 9 éléments : le premier élément représente la position du pavé manquant, le deuxième élément la position du pavé **1**, ... , le neuvième élément la position du pavé **8**.

   La position d'un pavé sera représentée par un couple de coordonnées X_i/Y_i .





Nœud courant

Nœud courant : [2/2,3/1,3/2,3/3,2/1,1/1,2/3,1/3,1/2]

(a) Définir le prédicat **manhattan(X1/Y1,X2/Y2,D)** vrai si et seulement si **D** est la distance "manhattan" entre les pavés de coordonnées **X1/Y1** et **X2/Y2**.

La distance "manhattan" est la somme des distances horizontale et verticale entre les pavés **X1/Y1** et **X2/Y2** :

$$D = \mid X_2 - X_1 \mid + \mid Y_2 - Y_1 \mid$$

(b) Définir le prédicat **successeur(+Noeud,?Noeud1,?N)** vrai si et seulement si **Noeud1** représente le nœud du graphe déduit du **Noeud** précédent par échange du pavé manquant avec l'un de ses voisins immédiats (**N** = 1 dans tous les cas).

(c) Définir le prédicat **estim(+Noeud,+NoeudArrivee,?N)** vrai si et seulement si **N** est le nombre de coups estimés pour passer du **Noeud** courant au **NoeudArrivee** particulier [**2/2,1/3,2/3,3/3,3/2,3/1,2/1,1/1,1/2**].

On utilisera la formule "magique"

$$\mathbf{N = DT + 3S}$$

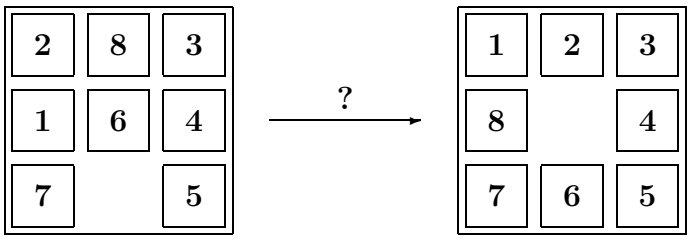
où

- **DT** est la distance "manhattan" totale entre le **Noeud** courant et le **NoeudArrivee** particulier [**2/2,1/3,2/3,3/3,3/2,3/1,2/1,1/1,1/2**] .
- **S** mesure à quel point les pavés sont déjà en place par rapport à la configuration finale recherchée.
  S est la somme des "scores" de tous les pavés déterminés suivants les règles suivantes :
  - le pavé central a un "score" de 1 ;
  - un pavé non central a un "score" de 0 si son successeur se trouve juste après lui en tournant dans le sens des aiguilles d'une montre ;
  - tous les autres pavés ont un "score" de 2 .

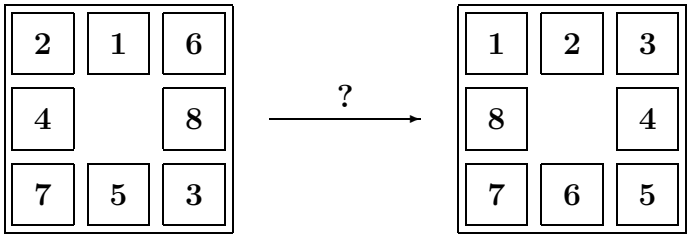
```
score(2/2,_,1) :- !.
score(1/3,2/3,0) :- !.
score(2/3,3/3,0) :- !.
score(3/3,3/2,0) :- !.
score(3/2,3/1,0) :- !.
score(3/1,2/1,0) :- !.
score(2/1,1/1,0) :- !.
score(1/1,1/2,0) :- !.
score(1/2,1/3,0) :- !.
score(_,_,2).
```

Bien que "magique", cette fonction heuristique ne garantit pas que le chemin optimal soit trouvé en premier !

(d) Tester les différents types de recherche du meilleur chemin dans les deux cas suivants :



Nœud initial **Nœud final**



Nœud initial **Nœud final**

5 Jeux avec adversaires

Un jeu est une suite de tours; à chaque tour, un joueur choisit un coup, exécute ce coup, et passe la main au joueur suivant. Etant donnés un joueur et un état du jeu, le problème consiste à trouver un chemin de coups vers un état gagnant.
On utilisera les prédicats suivants :

initialisationJeu(?EtatInitial,?PremierJoueur)

Initialise le jeu en fixant l'état initial (**EtatInitial**) du jeu et le premier joueur (**PremierJoueur**).

etatJeu(+Etat,+Joueur)

Informe sur l'**Etat** du jeu pour le **Joueur** considéré.

finJeu(+Etat,+Joueur,?Resultat)

Teste si l'**Etat** du jeu est un jeu gagnant pour le **Joueur** considéré et retourne un **Resultat**.

resultatJeu(+Resultat)

Informe sur le **Resultat** de la partie. **Resultat** est déterminé par le prédicat **finJeu/3**.

coupSuivant(+Etat,+Joueur,?Coup)

Détermine le "meilleur" **Coup** à jouer pour le **Joueur** considéré dans l'**Etat** donné.

etatSuivant(+Etat,+Coup,?EtatSuivant)

Détermine le nouvel état (**EtatSuivant**) à partir du précédent **Etat** et du **Coup** joué.

joueurSuivant(+Joueur,?JoueurSuivant)

Détermine le nouveau joueur (**JoueurSuivant**) compte–tenu du précédent **Joueur**.

Compte–tenu des définitions précédentes, les prédicats **jeu/0** et **jeu/3** fournissent un cadre général pour des programmes de jeux à plusieurs joueurs.

```
jeu :-
     initialisationJeu(EtatInitial,PremierJoueur),
     etatJeu(EtatInitial,PremierJoueur),
     jeu(EtatInitial,PremierJoueur,Resultat).

jeu(Etat,Joueur,Resultat) :-
     finJeu(Etat,Joueur,Resultat),
     !,
     resultatJeu(Resultat).

jeu(Etat,Joueur,Resultat) :-
     coupSuivant(Etat,Joueur,Coup),
     etatSuivant(Etat,Coup,EtatSuivant),
     etatJeu(EtatSuivant,Joueur),
     joueurSuivant(Joueur,JoueurSuivant),
     !,
     jeu(EtatSuivant,JoueurSuivant,Resultat).
```

L'arbre de jeu est en général beaucoup trop important pour être exploré de façon exhaustive. Le prédicat **coupSuivant/3** devra donc reposer sur une heuristique. On supposera que les prédicats suivants sont déjà définis :

coupSuivant(+Etat,?Coup)

Détermine un **Coup** possible à partir de l'**Etat** considéré.

etatSuivant(+Etat,+Coup,?EtatSuivant)

Détermine le nouvel état (**EtatSuivant**) à partir du précédent **Etat** et du **Coup** joué.

estim(+Etat,?Valeur)

détermine une estimation de la "**Valeur**" d'un **Etat** donné par rapport à un jeu gagnant.

5.1 Procédure minimax

La procédure minimax est une méthode classique pour déterminer la "valeur" d'un état; elle est fondée sur l'exploration de l'arbre de jeu plusieurs coups à l'avance. Nous considérons dans la suite qu'il n'y a que deux joueurs, respectivement appelés le "joueur" et l' "adversaire".
Cette procédure suppose que lorsqu'il est confronté à plusieurs choix, l'adversaire fait le meilleur choix pour lui — c'est–à–dire le plus mauvais choix pour le joueur. Le but du joueur est alors de choisir un coup qui maximise la "valeur" de l'état après que l'adversaire ait exécuté son meilleur coup — c'est–à–dire celui qui a minimisé la "valeur" de l'état pour le joueur. D'où le nom minimax : ce qui est bon pour le joueur doit être mauvais pour l'adversaire, et inversement.

Cette stratégie se fait en tenant compte des différentes possibilités plusieurs coups à l'avance.
Définir les prédicats suivants :

1. **minimax(+N,+Etat,+MaxMin,?Coup,?Valeur)**

   vrai si et seulement si **Coup** est le coup de **Valeur** la plus élevée à partir de l'**Etat** donné, en explorant **N** coups à l'avance; **MinMax** est un indicateur qui précise si on maximise le coup (**MinMax = 1**) ou si on le minimise (**MinMax = -1**).

2. **coupSuivant(+N,+Etat,+Coups,+MinMax,+MCoup,?MeilleurCoup)**

   vrai si et seulement si **MeilleurCoup** est le meilleur coup à jouer parmi la liste des coups possibles (**Coups**) dans l'**Etat** considéré, en explorant **N** coups à l'avance; **MinMax** est un indicateur qui précise si on maximise le coup (**MinMax = 1**) ou si on le minimise (**MinMax = -1**); **MCoup** est le meilleur coup déjà trouvé au moment de l'appel.

   MCoup et **MeilleurCoup** seront représentés par des doublets **(Coup,Valeur)**. Au premier appel **MCout** sera de la forme **('?',Min)**, où **'?'** représente n'importe quel coup et **Min** est une valeur inférieure à toute valeur possible de la fonction d'estimation **estim/2**.

5.2 Elagage $\alpha - \beta$

La procédure minimax peut être améliorée en sauvegardant une trace des résultats de l'exploration de l'arbre de jeu. L'idée est de conserver pour chaque nœud la valeur estimée minimum déjà rencontrée — la valeur α — et la valeur estimée maximum déjà rencontrée — la valeur β . Si, lors de l'évaluation d'un nœud, on dépasse la valeur maximum β, l'exploration de cette branche peut être annulée. D'où le nom d'élagage α – β.

5.3 Jeu de Kalah

Le jeu de Kalah se joue à deux sur un plateau composé de deux rangées de 6 trous. Chaque joueur possède une rangée de 6 trous, plus un trou (appelé kalah) situé à droite de sa rangée. Dans l'état initial, chaque trou contient 6 pions, et les 2 kalahs sont vides.
Un joueur commence la partie en ramassant tous les pions de l'un de ses trous (au libre choix du joueur).Puis, en progressant dans le sens contraire des aiguilles d'une montre, le joueur met un pion par trou, y compris dans sa kalah (mais pas dans celle de son adversaire), jusqu'à épuisement des pions ramassés.
Quatre attitudes sont alors possibles :

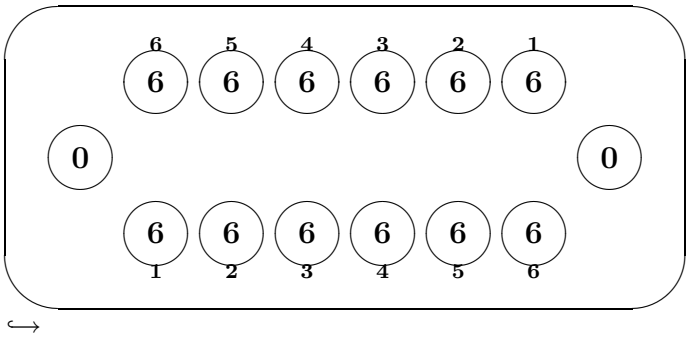
1. si le dernier pion tombe dans sa kalah, le joueur rejoue;

2. si le dernier pion tombe dans un trou vide de sa rangée, et si le trou de son adversaire directement opposé n'est pas vide, le joueur ramasse tous les pions du trou opposé et le dernier pion joué, et les met dans sa kalah;

3. si tous les trous d'un joueur deviennent vides (même si ce n'est pas son tour), les pions restés dans la rangée de son adversaire sont transférés dans la kalah de l'adversaire;

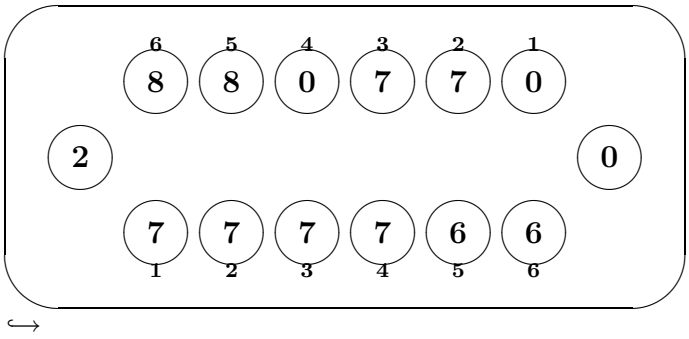
4. dans tous les autres cas, le joueur passe la main à son adversaire.

Le vainqueur est le premier des joueurs à avoir plus de la moitié des pions dans sa kalah.



Etat initial

coup joué : [1,4]



Etat suivant