Département Informatique - Cycle Ingénieur : Deuxième Année

Programmation par contraintes T.P. N^o 1

30 mars 2009

Intégration des Arc-consistances dans les algorithmes de recherche - Application : les quatre/huit reines

1 Le parcours en Profondeur d'abord

Nous présentons un programme prolog qui effectue la recherche en profondeur d'abord. Ce programme est basé sur a récursivité. Le prédicat solve permet de retrouver l'état final à partir d'un état donné. Les états déjà visités sont stockés dans une liste. Le prédicat *move* associe des actions à des états de départs. Le prédicat *update* change l'état en appliquant une action. Finalement le prédicat *legal* teste si un état existe ou non. Le programme en prolog est donné par :

```
solve(State,History,[]) :-etat_final(State).
solve(State,History,[Move|Moves]) :-
move(State,Move), update(State,Move,State1), legal(State1), not member(State1,History), solve(State1,[State1|History],Moves).
test(Moves) :-etat_initial(State), solve(State,[State],Moves).
```

2 Le parcours en Largeur d'abord

Ce programme est composé de trois prédicats :

solveB effectue l'exploration en largeur et s'arrête dès qu'il trouve une solution ;

update-set Met à jour la liste des noeuds à explorer en remplacent le premier élément par ses fils (ces fils sont insérés à la fin) ;

insertF insert un élément à la fin d'un liste donné si jamais cet élément correspond à un noeud non visité auparavant.

```
solveB([state(State,Path)|Reste],_,Moves) :-
etat_final(State), reverse(Path,Moves).
solveB([state(State,Path)|Reste],History,Finalpath) :-
not(etat_final(State)), findall(M,move(State,M),Moves),
update-set(Moves,State,Path,History,Reste,Reste1), solveB(Reste1,[State|History],Finalpath).
```

```
update-set([M|Ms],State,Path,History,R,R1) :-
update(State,M,State1), insertF(state(State1,[M|Path]),History,R,R2), update-set(Ms,State,Path,History,
update-set([],S,P,H,R,R).
insertF(state(State,_),History,[],[]) :-
member(State,History), !, insertF(X,History,[],[X]).
insertF(X,History,[T|Reste],[T|Reste1]) :-
insertF(X,History,Reste,Reste1).
test(Moves) :-etat_initial(State), solveB([state(State,[])],[State],Moves).
```

2.1 Rappel

Nous rappelons l'implémentation que nous avons proposé pour résoudre le problème de huit reines :

```
form(N,L,N) :-member(N,L).
form(I,L,N) :-I<N,member(I,L),I1 is I+1,form(I1,L,N).
queen(L) :-length(L,8),form(1,L,8),safe(L).
safe([]).
safe([X|L]) :-noattack(L,X,1),safe(L).
noattack([],_,_).
noattack([Y|L],X,I) :-diff(X,Y,I),I1 is I+1, noattack(L,X,I1).
diff(X,Y,I) :-X=\= Y, X =\= Y + I, X + I =\= Y.
```

Exercice 1 Appliquer le parcours en profondeur sur le problème des quatre reines.

Exercice 2 Intégrer l'arc consistance au niveau de chaque état de recherche, comparer les performances des deux algorithmes.

Exercice 3 Appliquer le parcours en largeur sur le problème des quatre reines.

Exercice 4 Intégrer l'arc consistance au niveau de chaque état de recherche, comparer les performances des deux algorithmes.