University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 5

Agenda Based Search

- Shortcomings of Algorithm
- 3 improvements

Department of Computer Science and AI

1998/9

The Program
Main Algorithm

```
agenda_search(A,Ans) :-
   agenda_get(A,N,Anew),
   agenda_search_1(N,Anew,Ans).

agenda_search_1(N,_,N) :-
   node_test(N),
   !.       % cut inhibits next clause
            % on success

agenda_search_1(N,A,Ans) :-
   node_expand(N,NodeList),
   agenda_putl(A,NodeList,NewA),
   agenda_search(NewA,Ans).
```

Agenda Primitives

```
agenda_new([]).

agenda_get([X|Y],X,Y).

% With this definition
% we obtain breadth first search

agenda_putl(A,L,Anew) :- append(A,L,Anew).

agenda_init(Ainit) :-
    agenda_new(A),
    node_initial(N),
    agenda_putl(A,[N],Ainit).
```

Problem Specification

```
stn(bs,[gp,oc]).
stn(gp,[bs,cc,pc]).
stn(cc,[gp,ls,pc]).
stn(pc,[cc,gp,ls,oc]).
stn(ls,[cc,pc,tc]).
stn(tc,[ls,oc]).
stn(oc,[bs,pc,tc]).

node_initial(bs). % start node

node_test(cc).    % goal node

node_expand(N,L) :- stn(N,L).

solve(Ans) :-
  agenda_init(A), agenda_search(A,Ans).

?- solve(Ans).
Ans = cc ;
No
```

Shortcomings of agenda_search version 1

- Start and end-points of search are wired in. We should be able top specify them as part of the query.
- Too little information
  - Nodes examined
  - Efficiency of search
  - Path to goal
  - Quality of solution
- Fixed search strategy.

Parameterising Initial and Goal States

Currently, initial and goal states are directly encoded into the database as clauses

```
% start node
node_initial(bs).


% test for goal
node_test(N).
%solving predicate
```

This means that to change the problem, we must modify the database. It would be much more convenient to have "Initial" and "Goal" as parameters to the solving procedure, i.e.

```
solve(Start,Goal) :- ....
```

Building solve(Start,Goal)

There are basically two ways to do this:

1. **Extra Parameters.** These are passed around by the solving procedure. Advantage: declarativity. Disadvantage: readability.
2. **Using the Database.** We can change the definitions of node_initial and node_test on the fly. Advantage: main program changes minimally and retains its readability.

2. Database Solution the built-in abolish

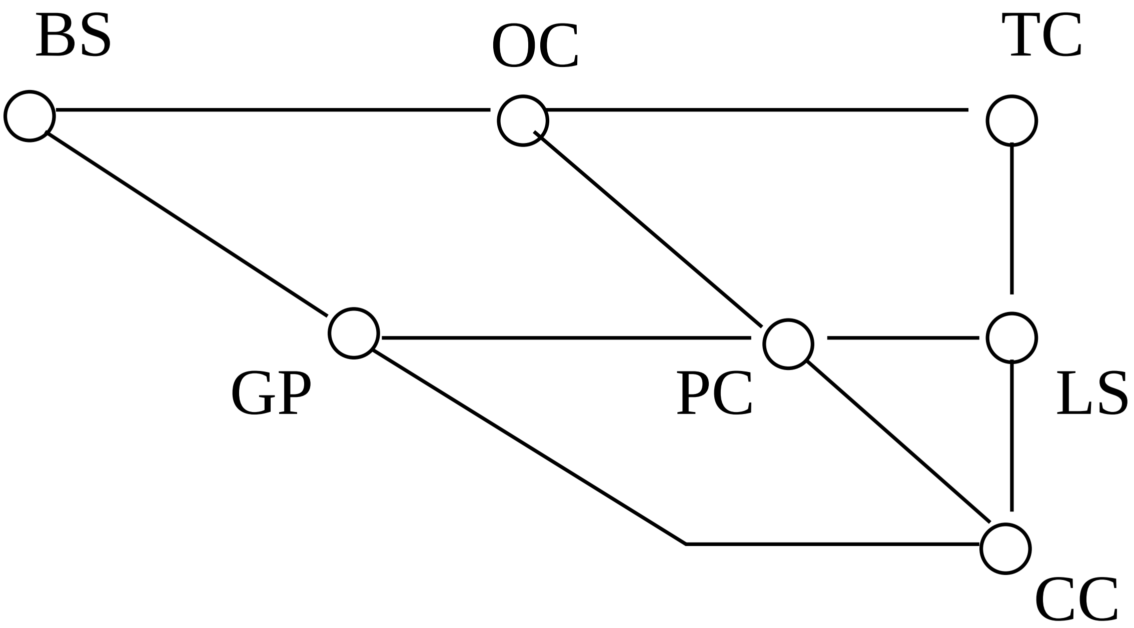
N.B. The predicate abolish(P,N) erases the database definition of P/N (i.e. P with N arguments).

```
start_init(S) :-
 abolish(node_initial,1),
 asserta(node_initial(S)).

goal_init(S) :-
 abolish(node_test,1),
 asserta(node_test(S)).
```

To make use of these new definitions, we need to incorporate them into the definition of solve.

Result



?- solve(bs,cc,Z).

Z = cc ;

Display of Information
Nodes Visited

Currently, we get no information concerning the progress of the search. A very simple improvement is to simply print out each and every node that is processed.

```
node_display(N) :- write(N), nl.
```

The question is, where to put it. Here is the result if it is put in the right place:

```
?- solve(bs,cc,Ans).
bs
gp
oc
bs
cc

Ans = cc ;

No
```

Efficiency of Search

Efficiency of search is directly related to the number of nodes examined. To extract information about efficiency, we therefore need to do some counting.

Again we have several choices about how to do this. Possibilities include:

1. via extra parameters
2. via the database
3. by tagging each search node with a unique number.

Below we consider the first solution.

Using Extra Parameters to Count Nodes

The basic idea here is to give solve one extra parameter, which will be the number of nodes examined. This will then be passed on to **agenda_search**.

```
solve(Start,Goal,Ans,N) :-
 ...
 agenda_search(0,N,A,Ans).
```

agenda_search itself requires 2 extra integer parameters, namely, an input node number, which is initialised to 0, and N, the output node number, which will be passed back to the query:

```
?- solve(bs,ls,Ans,N).
bs ... ls
N = 13 ?
```