University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 7

Agenda Based Search

- Simulating Variables
- Generic Updating of the Agenda
- Including Depth Information in Nodes
- Best First Search

Department of Computer Science and AI
1998/9

Simulating Variables

One way to avoid cluttering up the general search algorithm with extra arguments is to create "variables". These can be used, for example

- To store information about start and goal states.
- To store the number of nodes created.

Behaviour of Variables

```
?- initv(counter), getv(counter,Z).
Z = 0

?- inc(counter), getv(counter,Z).
Z = 1

?- dec(counter), getv(counter,Z).
Z = 1

?- setv(counter,10), getv(counter,Z)
Z=10

?- setv(pred,silly(sally)), getv(pred,Z).
Z=silly(sally)
```

Code For Simulating Variables

```
setv(V,VAL) :-                % Set V to VAL
  retractall(isvar(V,_)),
  asserta(isvar(V,VAL)).

getv(V,VAL) :-                % Retrieve value of V
  isvar(V,VAL),
  !.

initv(V) :-                   % Initialise a counter
 setv(V,0).

inc(V) :-                     % increment V
 getv(V,N),
 N1 is N+1,
 setv(V,N1).

dec(V) :-                     % decrement V
 getv(V,N),
 N1 is N-1,
 setv(V,N1).
```

Reducing Clutter: Example

- Before

```
agenda_search(EXCL,C,Cfinal,A,G) :-
   agenda_get(EXCL,C,C1,A,N,Anew),
   agenda_search_1(EXCL,C1,Cfinal,N,Anew,G).
```

- After

```
% set up node counter variable
initvar(nodes_created).

% increment it when you create new nodes
node_new(n(_,_,_,_)) :- inc(nodes_created).

% now all node counting args can be
% removed from main routines
agenda_search(EXCL,G) :-
   agenda_get(EXCL,A,N,Anew),
   agenda_search_1(EXCL,N,Anew,G).
```

Adding Depth Information to Nodes

- **Depth First Search**: maintain an agenda with nodes in decreasing order of depth.

  Currently, DFS implemented by putting newly expanded nodes at the **front** *of the agenda (using append)*

- **Breadth First Search**: maintain an agenda with nodes in increasing order of depth.

  Currently, BFS implemented by putting newly expanded nodes at the **back** *of the agenda (still using append, but with a different order of arguments).*

- **Problem**: How do we handle **Best First Search** using this scheme?

- **Answer**: Use a more flexible strategy for updating the agenda.

Finer Control over Strategy for Updating Agenda

- Assume existence of a predicate **node_eval(N,V)** which assigns a value V to node V.
- Assume existence of a partial ordering relation amongst nodes such that at least one of the following are defined:
  - **node_gt(N1,N2)**: succeeds if N1 is "greater than" N2 in the ordering.
  - **node_leq(N1,N2)**: succeeds if N1 is "less than or equal to" N2 in the ordering.
- Define a general relation

  add_generic(Agenda,Childen,NewAgenda)

  which updates the Agenda with Children to form NewAgenda according to the above ordering.

Defining add_generic

```
% add_generic(OldAgenda,Nodes,NewAgenda).
% adds Nodes to OldAgenda to form Newagenda
% according to "node_leq" ordering relation
% OldAgenda is assumedto be ordered already.

add_generic(A,[],A).

add_generic(OldA,[N|RestN],NewA) :-
 add1_generic(N,OldA,TmpA),
 add_generic(TmpA,RestN,NewA).

add1_generic(N,[],[N]).

add1_generic(N,[Old1|OldRest],[N|[Old1|OldRest]]) :-
 node_leq(N,Old1),
 !.

add1_generic(N,[Old1|OldRest],[Old1|NewRest]) :-
 add1_generic(N,OldRest,NewRest).
```

Behaviour of add_generic

For this to worl node_leq has to be defined.

```
?- add_generic([1,2,3,4],[5,6,7,8],Z).
Z = [1, 2, 3, 4, 5, 6, 7, 8] ;

?- add_generic([1,2,3,4],[8,7,6,5],Z).
Z = [1, 2, 3, 4, 5, 6, 7, 8] ;

?- add_generic([4,3,2,1],[5,6,7,8],Z).
Z = [4, 3, 2, 1, 5, 6, 7, 8] ;

?- add_generic([5,6,7,8],[1,2,3,4],Z).
Z = [1, 2, 3, 4, 5, 6, 7, 8] ;
```

Reconstructing DFS and BFS

To reconstruct DFS and BFS we must now

- Maintain appropriate depth information in nodes
- Define node_eval.

```
node_eval(N,V) :- node_depth_of(N,V).
```

- Define node_leq

```
node_leq(N1,N2) :-
 node_eval(N1,V1),
 node_eval(N2,V2),
 V2 > V1.
```

Checkpoint. This relation ranks the agenda in order of increasing depth. Does this correspond to DFS or BFS?

How would you get the opposite strategy?

Adding Depth Information to Nodes

There are 2 places in the program where depth information must be added

- When the initial node is created, we know the depth is zero.
- Whenever we expand a node, then the depth of a child node is always one greater than the parent

Code for Adding Depth Information

```
node_initial(N) :-
 start_state(S),
 node_new(N),
 node_state_of(N,S),
 node_parent_of(N,-1),
 node_depth_of(N,0).     <----


sl2nl(P,[S|Y],[N|NL]) :-
 node_new(N),
 node_state_of(N,S),
 node_parent_of(N,P),
 node_depth_of(P,PD),     <----
 PD1 is PD+1,             <----
 node_depth_of(N,PD1),    <----
 sl2nl(P,Y,NL).
```

Best First Search

Best first search is just like breadth first search except that instead of using a depth measure to order nodes in the agenda, we use a cost function, f.

The agenda must be maintained in order of increasing cost.

There are various ways of defining f out of two other components, g(N), the path cost, and h(N), the heuristic cost.

- f(N) = g(N), gives us **uniform cost search**.
- f(N) = h(N) gives us **greedy** search.
- f(N) = g(N) + h(N) gives us **A* search**, which is guaranteed to be complete and optimal if f is admissible (never overestimates the true cost).