IDA* in Depth

Not in Norvig&Russell book

Source of Code can be found in “Computational Intelligence”

By Poole, Mackworth, & Goebel

Outline

- What we need
- F-Bounded Search
- IDA* Search

Need Defined

Domain definitions:

----

neighbors(State, Neighbors)
cost(State, Neighbor, ArcCost)

Problem definition:

----

is_goal(State)

Search definition:

----

h(State, HeuristicValue)

F-Bounded Search

- *fbsearch(Frontier,FBound,Q,NextFBound,Path)*
- “*Frontier*” is treated as a stack (we’re just doing f-bounded depth-first search).
- “*FBound*” is the f-bound for this iteration.
- “*Q*” is the Frontier with just the initial state node, used when starting a new iteration.
- “*NextFBound*” keeps track of the next iteration’s FBound.
- “*Path*” is the path from the root to the head of Frontier, and is used to return the solution.

F-Bound Calculations

- For the initial iteration, the f-bound is simply the h-value of the intial state.
- For subsequent iterations, the new f-bound is least f-value of the previous iteration that exceeded its f-bound.

F-Bounded Search

- There are the following cases for an f-bounded search, head of Frontier (*node*):
  - *node* is a goal node: found solution.
  - Frontier is empty, done with this iteration
    - If there were some nodes whose f-values exceeded the current f-bound, start next iteration.
    - If there were no nodes whose f-values exceeded the current f-bound, no solution exists.
  - Otherwise: pop head(Frontier) &
    - If *f(node)* =< *Fbound:* push neighbors of node onto Frontier & continue
    - Otherwise *f(node)* > *Fbound:* continue

F-Bounded Search

Head of Frontier is a goal node: found solution

```
fbsearch([node(State,Path,FBound)| _ ],
              FBound, _, _, [State | Path])     :-
         % we found a solution
         is_goal(State).
```

F-Bounded Search

Frontier is empty: done with this iteration, start next iteration.

```
fbsearch([ ],_, Q, NextFBound, Solution) :-
        % finished searching at this f-Bound
        % and we actually expanded some nodes
        % start searching at the next f-Bound
        initialNextFBound(InitialNextFBound),
        InitialNextFBound > NextFBound,
        writeln(['Trying Depth bound: ',NextFBound]),
        fbsearch(Q, NextFBound,
                        Q, InitialNextFBound,Solution).
```

F-Bounded Search

Frontier is empty: done with this iteration, start next iteration.

- If *NextFBound* = *InitialNextFBound* then either the least next F Bound was
  - 10,000 or no nodes’s f-values exceeded the current iteration’s F Bound

F-Bounded Search

If *f(node) =< Fbound*

```
fbsearch([node(State,Path,PathCost) | OldFrontier],
                        FBound, Q, NextFBound, Solution) :-
          h(State,HeuristicValue),
          FValue is HeuristicValue + PathCost,
          FBound >= FValue,
          % f-Bound >= fValue
          % we pop this node off of frontier
          % we expand this node & push its children onto frontier
          neighbours(State, Neighbors),
          add_paths_fb(Neighbors, State, [State | Path],
                                 PathCost, OldFrontier, NewFrontier),
          fbsearch(NewFrontier, FBound, Q,
                         NextFBound, Solution).
```

F-Bounded Search

If *f(node) > Fbound*

```
fbsearch([node(State,_,PathCost) | Frontier], FBound, Q,
            NextFBound, Solution) :-
        h(State, HeuristicValue),
        FValue is HeuristicValue + PathCost,
        FValue > FBound,
        % fValue > f-Bound
        % we pop this node off of frontier
        % don't expand this node
        % see if its fValue will be the next f-Bound
        LeastUpperBound is min(FValue, NextFBound),
        fbsearch(Frontier, FBound, Q, LeastUpperBound,
                   Solution).
```

IDA* Top Level

```
idaStarSearch(State,Solution) :-
        h(State,HeuristicValue),
        FBound = HeuristicValue,
        writeln(['Trying f-bound: ', FBound]),
        initialNextFBound(NextFBound),
        fbsearch([node(State,[ ],0)],
                    FBound,
                    [node(State,[ ],0)],
                    NextFBound,
                    Solution).
```

add_paths_fb(Neighbors, State, Path, OldPathCost, OldFrontier, NewFrontier)

```
/* Neighbors are the states neighboring State, we turn each*/
/*neighboring state into a node, and push them onto the frontier.*/
/*Nodes contain the state, the path from the root to that state, */
/* and the cost of that path.*/
add_paths_fb([ ],_,_,_,Frontier,Frontier).
add_paths_fb([Neighbor  | Rest], State, Path,
                              OldPathCost, OldFrontier,
                     [node(Neighbor,Path,NewPathCost)
                      | NewFrontier]) :-
          cost(State,Neighbor,ArcCost),
          NewPathCost is OldPathCost + ArcCost,
add_paths_fb(Rest, State, Path, OldPathCost,
                              OldFrontier, NewFrontier).
```