6.034 Recitation 2
Search Methods: Addendum

Luis E. Ortiz

9/24/04

1 Basic Search

Algorithm 1 is the basic (template) pseudocode for the different basic search methods. Recall that for a partial path N, $head(N)$ is the first element in the list representation of the partial path, which corresponds to the last node in the path that starts at S.

Algorithm 1 Basic Search

1: Initialize $Q = ((S))$
2: **while** Q is not empty **do**
3: Pick *some* partial path N from Q, and remove it from Q
4: **if** $head(N)$ is the goal node **then**
5: **return** N
6: **end if**
7: **for all** descendants d of $head(N)$ in the search tree **do**
8: expand the node to extend the partial path N to d
9: **end for**
10: *Add* extensions of N *somewhere* in Q
11: **end while**
12: Signal failure

Look at one of the handouts given in the recitation with the table stating how to instantiate the steps 3 and 10; that is, (1) which partial path N to pick from the queue and (2) where to insert the extended paths in the queue. The same applies to the pseudocode that follows below.

Algorithm 2 shows the pseudocode for basic search with an enqueued list. The differences from the basic search pseudocode above (without an enqueued list) are in italics.

Recall that beam search is slightly different from the basic search pseudocode above in that we remove and extend every path in the queue, and then add all the new extended paths to the queue and keep only the best k in the queue (and remove the rest). Algorithm 3 is

Algorithm 2 Basic Search *with Enqueued List*

```
1: Initialize Q = ((S)) and E = (S)
2: while Q is not empty do
3:    Pick some partial path N from Q, and remove it from Q
4:    if head(N) is the goal node then
5:       return N
6:    end if
7:    for all descendants d of head(N) in the search tree with d not in E do
8:       expand the node to extend the partial path N to d and add d to E
9:    end for
10:   Add extensions of N somewhere in Q
11: end while
12: Signal failure
```

the version of beam search with an enqueued list (the version without an enqueued list is similar).

Algorithm 3 Beam Search with Width k *and Enqueued List*

```
1: Initialize Q = ((S)) and E = (S)
2: while Q is not empty do
3:    for all partial paths N in Q do
4:       if head(N) is the goal node then
5:          return N
6:       end if
7:       for all descendants d of head(N) in the search tree with d not in E do
8:          expand the node to extend the partial path N to d and add d to E
9:       end for
10:   end for
11:   Replace Q with (at most) the best k extensions of N
12: end while
13: Signal failure
```

The difference between *informed or heuristic* and *uninformed or blind* search is that in informed search, for each node n, we assign to n a heuristic value that is an estimate of the minimum number of steps to the goal from n, and use those heuristic values to decide which node to expand next (i.e., which partial path in the queue to pick next). In general, we want to pick paths whose head (i.e., last node in the path) has low heuristic value. Depth-first and breadth-first search are uninformed search methods, while hill climbing, best-first and beam search are informed search methods.

Finally, the *British Museum* algorithm is a *brute-force* search method in that it considers *every* possible path in the search tree before it selects the best path. As such, it does not conform to the pseudocode given above.

1.1 Other considerations

One of the handouts given in class contains a table giving *worst-case time and space bounds* for the different basic search algorithms. These bounds are given in terms of the *branching factor* and *depth* of the search tree. The branching factor is the maximum number of descendants of any node in the search tree and is assumed constant for the purpose of the analysis. The *number of expanded nodes* during the execution of the algorithm characterize the *worst-case running time* of the different basic search algorithms. Similarly, the *space complexity* is characterized by the *maximum size of the queue* (i.e., the number of elements or partial paths) at any time during the execution of the algorithm.

Note also that only breadth-first search is guaranteed to find a path to the goal with the minimum number of nodes. Breadth-first search is always guaranteed to find a path to the goal (i.e., it is complete), even if the search tree is infinite. For search trees of finite depth, depth-first search and hill climbing *with backup* as well as best-first search are also guaranteed to find a path to the goal.

2 Optimal Search

Algorithm 4 is the pseudocode for the optimal search algorithms based on branch and bound. The version given uses an extended list. An extended list is different from the enqueued list used for the basic search algorithms. The extended list keeps tracks of which nodes have been expanded/extended during the execution of the algorithm; even if the node is *enqueued* several times, once it has been expanded/extended and inserted into the extended list, it will not be extended again. You can obtain the optimal search pseudocode without an extended list by removing the parts in italics involving the variable E.

Just as in informed basic search, we can use a heuristic function $h(n)$ over the nodes n of the graph as an estimate of the cost to the goal from n. A heuristic h is *admissible* if it *does not overestimate the cost to the goal* (i.e., for all nodes n, if $c(n)$ is the cost of the path with minimum cost to the goal from n, then $h(n) \leq c(n)$), and it *satisfies a kind of triangle inequality* (i.e., for all nodes n and all its descendants m, if $d(n, m)$ is the cost of the path from n to m, then $h(n) \leq d(n, m) + h(m)$). The basic intuition about admissible heuristics is that they do not allow us to miss a shorter path to the goal because we have overestimated its costs due to an overestimate in one of the nodes in the optimal path.

There are different versions of branch and bound for optimal search depending on whether or not one uses an admissible heuristic and whether one uses an extended list. The A^* algorithm is simply branch and bound with an admissible heuristic and an extended list.

The instantiation of which partial path N to pick in step 3 of the pseudocode depends on which version of branch and bound we are using, but independent of whether we are using an extended list or not. For branch and bound *without and admissible heuristic*, we pick the path with the minimum cost so far. For branch and bound *with an admissible heuristic*, we pick the path with the minimum sum of the cost so far and the heuristic cost estimate to the goal.

Algorithm 4 Optimal Search *with Extended List*

Initialize $Q = ((S))$ *and* $E = ()$
while Q is not empty **do**
  Pick *some* partial path N from Q, and remove it from Q
  if $head(N)$ is the goal node **then**
    return N
  end if
  if $head(N)$ *in* E ***then***
    continue *(i.e., do not extend* N *and instead go back to the top of while loop)*
  end if
  Add $head(N)$ *to* E
  for all descendants d of $head(N)$ in the search tree **do**
    expand the node to extend the partial path N to d
  end for
  Add extensions of N *anywhere* in Q
end while
Signal failure