Artificial Intelligence
Search II
Lecture 4

Constrained Satisfaction Search (cont)

Example: 8-Queens Problem

- **States**: \(<V_1, V_2, ..., V_8>\), where \(V_i\) is the row occupied by the \(i\)th queen.
- **Domains**: \{1, 2, 3, 4, 5, 6, 7, 8\}
- **Constraints**: no-attack constraint
  \{<1,3>, <1,4>, <1,5>, ..., <2,4>, <2,5>, ...\}
  where each element specifies a pair of allowable values for variables \(V_i\) and \(V_j\).

Constrained Satisfaction Search (cont)

- **Types of Constraints**
  - Discrete (e.g. 8-queens) versus Continuous (e.g. phase I simplex).
  - Absolute constraints (violation rules out solution candidate) versus preferential constraints (e.g. goal programming).
- **Other better search strategies**
  - Backtracking search, forward checking, arc consistency checking.
  - Constraint propagation.

Heuristic Search Methods

- It uses heuristics, or rules-of-thumb to help decide which parts of a tree to examine.
- A heuristic is a rule or method that almost always improves the decision process.

Definition of a Heuristic Function

The value refers to the cost involved for an action. A continual based on \(H(s)\) is 'heuristically' the best.
Heuristics for an 8-puzzle Problem

Two possible heuristic functions that never overestimate the number of steps to the goal are:

1. $h_1$ = the number of tiles that are in the wrong position. In figure 5.7, none of the 8 tiles is in the goal position, so that start state would have $h_1 = 8$. $h_1$ is admissible heuristic, because it is clear that any tile that is out of the place must be moved at least once.

2. $h_2$ = the sum of distance of the tiles from their goal positions. Since no diagonal moves is allow, we use Manhattan or city block distance. $h_2$ is also admissible, because any move can only move 1 tile 1 step closer to the goal. The 8 tiles in the start state give a Manhattan distance of $h_2 = 2+3+3+2+4+2+0+2 = 18$.

Best First Search - An example

For sliding tiles problem, one suitable function is the number of tiles in the correct position.

Best First Search

1. open = [A5]; closed = []
2. evaluate A5; open = [B4,C4,D6]; closed = [A5]
3. evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]
4. evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]
5. evaluate H3; open = [C2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]
6. evaluate C2; open = [F3,G4,E5,F5,D6]; closed = [C2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!
Heuristics for an 8-puzzle Problem (cont)

Figure 5.8 State space generated in heuristics search of a 8-puzzle.

\[ f(n) = g(n) + h(n) \]

- **g(n)**: actual distance from start to state **n**
- **h(n)**: number of tiles out of place

Best First Search

1. Operations on states generate children of the state currently under examination.
2. Each new state is checked to see whether it has occurred before (is on either open or closed), thereby preserving loops.
3. Each state \( n \) is given an \( f \) value equal to the sum of its depth in the search space \( g(n) \) and a heuristic estimate of its distance to a goal \( h(n) \). The \( h \) value guides search toward heuristically promising states while the \( g \) value prevents search from persisting indefinitely on a futile path.
4. States on open are sorted by their \( f \) values. By keeping all states on open until they are examined or a path is found, the algorithm can go back from fruitless paths. At any one time, open may contain states at different levels of the search space graph, allowing full flexibility in changing the focus of the search.
5. The efficiency of the algorithm can be improved by careful maintenance of the open and closed lists, perhaps as a heap or leftist tree.
For this search, called the greedy (or best first) search, we assume the h values of all nodes to be zero. Then, for all nodes x, we obtain f(x) = g(x). From all the expanded nodes, we thus expand the node that has the lowest g value; that is, the node from which we incurred the lowest cost in traversing from the source node. If all the moved cost the same, then the nodes are expanded depth by depth, as in the uninformed breadth-first search of Section II.3.1. Incurred cost search is also known in the literature as the uniform cost search.

For this search, called the predicted cost search, we ignore the g values of all nodes. Then, for all nodes x, we obtain f(x) = h(x). From all the unexpanded nodes, we thus expand the node that has the lowest h value; that is, the node from which we expect the lowest cost in traversing to the destination node. The lowest literals preference strategy (Section 4.2.2) of resolution: relaxation exemplifies a predicted cost search for deducing the empty clause. Predicted cost search is also known in the literature as the greedy search.

BestFS1: Greedy Search

- This is one of the simplest BestFS strategies.
- Heuristic function: h(n) - prediction of path cost left to the goal.
- Greedy Search: "To minimize the estimated cost to reach the goal".
- The node whose state is judged to be closest to the goal state is always expanded first.
- Two route-finding methods (1) Straight line distance; (2) minimum Manhattan Distance - movements constrained to horizontal and vertical directions.

BestFS2: A* Search

- GS minimize the estimate cost to the goal, h(n), thereby cuts the search cost considerably - but it is not optimal and incomplete.
- UCS minimize the cost of the path so far, g(n) and is optimal and complete but can be very inefficient.
- A* Search combines both GS h(n) and UCS g(n) to give f(n) which estimated cost of the cheapest solution through n, ie f(n) = g(n) + h(n).
BestFS2: A* Search (cont)

✓ \( h(n) \) must be a admissible solution, i.e., it never overestimates the actual cost of the best solution through \( n \).

✓ Also observe that any path from the root, the f-cost never decreases (Monotone heuristic).

✓ Among optimal algorithms of this type - algorithms that extend search paths from the root - A* is optimally efficient for any given heuristics function.

A* search method (list implementation)

1. Initialize lists OPEN and CLOSED to empty.
2. Put the source node in OPEN.
3. Remove the front most node \( x \) from OPEN.
4. If \( x \) is a destination node, then return successfully with the solution path obtained by using the parentage pointers.
5. Put \( x \) in CLOSED.
6. Expand \( x \).
7. If no children of \( x \) are generated, that is, \( x \) is inert, then go to (3).
8. Delete all those children of \( x \) that are either renegade or recurring.

A* search method (list implementation) (cont.)

9. For every surviving child \( y \) of \( x \) do the following.
   9.1. If a copy of \( y \) is neither in OPEN nor in CLOSED, then put \( y \) in its earmarked position in OPEN.
   9.2. If a copy \( y_c \) of \( y \) is either in OPEN or CLOSED then do the following.
      9.2.1. If \( f(y) \geq f(y_c) \), that is, a more costly path through the node has been estimated, then delete \( y \).
      9.2.2. If \( f(y) < f(y_c) \), that is, a cheaper path through the node has been estimated, then delete \( y_c \), and put \( y \) in its earmarked position in OPEN (thus, if \( y_c \) was in OPEN, then only one copy of the node is retained in OPEN; if \( y_c \) was in CLOSED, then it is in effect transferred to OPEN so that later it may be expanded again).
10. Go to (2).

A* search:
- Form a one-element queue consisting of a zero-length path that contains only the root node.
- Until the first path in the queue terminates at the goal node or the queue is empty:
  - Remove the first path from the queue
  - Create new path by extending the first path to all the neighbors of the terminal node
  - Reject all new path with the loops
  - If two or more paths reach a common node, delete all those paths except the one that reaches the common node with minimum cost
  - Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least-cost paths in front.
- If the goal node is found, announce success, otherwise announce failure.

Figure 5.6 Map of Romania showing contours at \( f = 380 \), \( f = 400 \), and \( f = 420 \), with Arad as the start state. Nodes inside a given contour have f-costs lower than the contour value.
End of Lecture 4

Good Day.