4.0 Introduction
4.1 An Algorithm for Heuristic Search
4.2 Admissibility, Monotonicity, and Informedness
4.3 Using Heuristics in Games
4.4 Complexity Issues
4.5 Epilogue and References
4.6 Exercises
Heuristic Search

• “The study of the methods and rules of discovery and invention” defined by George Polya
• Greek root word is *euisco* which means “I discover”
• Archimedes emerged and shouted “Eureka” meaning “I have found it”
• In state space search, *heuristics* are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable solution
AI problem solvers employ heuristics in two basic situations:

- A problem **may not have an exact solution** because of inherent ambiguities in the problem statement or available data. A given set of symptoms may have several possible causes; doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment.

- A problem **may have an exact solution**, but the computational cost of finding it may be prohibitive. In many problems (such as chess), state space growth is combinatorially explosive, with the number of possible states increasing exponentially or factorially with the depth of the search. In these cases, exhaustive, brute-force search techniques such as depth-first or breadth-first search may fail to find a solution within any practical length of time. **Heuristics** attack this complexity by guiding the search along the most “promising” path through the space. By **eliminating unpromising states** and their descendants from consideration, a heuristics algorithm can defeat this combinatorial explosion and find an acceptable solution.
Inherent limitation of heuristic search

• Heuristic is only an informed guess of the next step to be taken in solving a problem

• Because heuristics use limited information, they are seldom able to predict the exact behavior of the state space farther along in the search

• A heuristic can lead a search algorithm to a sub-optimal solution or fail to find any solution at all
Importance of heuristics

• But heuristics and the design of algorithms to implement heuristic search have long been a core concern of artificial intelligence research.

• It is not feasible to examine every inference that can be made in a mathematics domain.

• Heuristic search is often the only practical answer.

• More recently, expert systems research has affirmed the importance of heuristics as an essential component of problem solving.
Figure 4.1: First three levels of the tic-tac-toe state space reduced by symmetry.
In tic-tac-toe problem,

- the combinatorics for exhaustive search are high but not insurmountable

- total number of states that need to be considered in an exhaustive search at $9 \times 8 \times 7 \times \ldots$ or $9!$

- Symmetry reduction decrease the search space

- Symmetry reductions on the second level further reduce the number of paths through the space to $12 \times 7!$ in the previous figure.
Figure 4.2: The “most wins” heuristic applied to the first children in tic-tac-toe.

Three wins through a corner square

Four wins through the center square

Two wins through a side square
Figure 4.3: Heuristically reduced state space for tic-tac-toe.
Hill climbing

- To implementation of heuristic search is through a procedure called *hill climbing*
- Hill climbing strategies expand the current state in the search and evaluate its children
- the best child is selected for further expansion; neither its siblings nor its parent are retained
- search halts when it reaches a state that is better than any of its children
- go uphill along the steepest possible path until it can go no farther
- because it keeps no history, the algorithm cannot recover from failures of its strategy
best-first search algorithm

• A major problem of hill climbing strategies is their tendency to become stuck at local maxima
• if they reach a state that has a better evaluation than any of its children, the algorithm halts
• hill climbing can be used effectively if the evaluation function is sufficiently informative to avoid local maxima and infinite paths
• heuristic search requires a more flexible algorithm: this is provided by **best-first search**, where, with a **priority queue**, recovery from local maxima is possible
function best_first_search;
begin
  open := [Start];
closed := [ ];
while open ≠ [] do % states remain
  begin
    remove the leftmost state from open, call it X;
    if X = goal then return the path from Start to X
    else begin
      generate children of X;
      for each child of X do case
        the child is not on open or closed:
          begin
            assign the child a heuristic value;
            add the child to open
          end;
        the child is already on open:
          if the child was reached by a shorter path
            then give the state on open the shorter path
        the child is already on closed:
          if the child was reached by a shorter path then
            begin
              remove the state from closed;
              add the child to open
            end;
          end;
      end; % case
      put X on closed;
      re-order states on open by heuristic merit (best leftmost)
    end;
  return FAIL % open is empty
end.
Figure 4.4: Heuristic search of a hypothetical state space.
A trace of the execution of `best_first_search` for Figure 4.4

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 = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]
6. evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!
• The best-first search algorithm always select the most promising state on open for further expansion.

• As it is using a heuristic that may prove erroneous, it does not abandon all the other states but maintains them on open.

• In the event a heuristic leads the search down a path that proves incorrect, the algorithm will eventually retrieve some previously generated, “next best” state from open and shift its focus to another part of the space.

• In best-first search, as in depth-first and breadth-first search algorithms, the open list allows backtracking from paths that fail to produce a goal.
Figure 4.5: Heuristic search of a hypothetical state space with open and closed states highlighted.
Figure 4.6: The start state, first set of moves, and goal state for an 8-puzzle instance.
Figure 4.8: Three heuristics applied to states in the 8-puzzle.

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 8 3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>1 6 4</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7 5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Tiles out of place
- Sum of distances out of place
- $2 \times$ the number of direct tile reversals

Goal:

- 1 2 3
- 8 4
- 7 6 5
Figure 4.9: The heuristic $f$ applied to states in the 8-puzzle.

Values of $f(n)$ for each state,

where:

- $f(n) = g(n) + h(n)$,
- $g(n)$ = actual distance from $n$ to the start state, and
- $h(n)$ = number of tiles out of place.