<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.1</td>
<td>An Algorithm for Heuristic Search</td>
</tr>
<tr>
<td>4.2</td>
<td>Admissibility, Monotonicity, and Informedness</td>
</tr>
<tr>
<td>4.3</td>
<td>Using Heuristics in Games</td>
</tr>
<tr>
<td>4.4</td>
<td>Complexity Issues</td>
</tr>
<tr>
<td>4.5</td>
<td>Epilogue and References</td>
</tr>
<tr>
<td>4.6</td>
<td>Exercises</td>
</tr>
</tbody>
</table>

George F Luger

**Artificial Intelligence**

Structure and Strategies for Complex Problem Solving

Fourth Edition
The successive stages of open and closed that generate this graph are:

1. open = [a4];
   closed = [ ]

2. open = [c4, b6, d6];
   closed = [a4]

3. open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]

4. open = [f5, h6, b6, d6, g6, l7];
   closed = [a4, c4, e5]

5. open = [j5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5]

6. open = [l5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5, j5]

7. open = [m5, h6, b6, d6, g6, n7, k7, l7];
   closed = [a4, c4, e5, f5, j5, l5]

8. success, m = goal!
Figure 4.10: State space generated in heuristic search of the 8-puzzle graph.
Figure 4.11: open and closed as they appear after the third iteration of heuristic search.
To summarize

- Operations on states generate children of the state currently under examination.
- Each new state is checked to see whether it has occurred before (is on either open or closed), thereby preventing loops.
- 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 fruitless path.
- States on open are sorted by their values. By keeping all states on open until they are examined or a goal is found, the algorithm recovers from dead ends.
- As an implementation point, the algorithm’s efficiency can be improved through maintenance of the open and closed lists, perhaps as heaps or leftist tree.
In general,

- Best-first search is a general algorithm for heuristically search any state space graph (as were in the breadth- and depth-first algorithms)

- It is equally appropriate data- and goal-driven searches and supports a variety of heuristic evaluation functions

- Because of its generality, best-first search can be used with a variety of heuristics, ranging from subjective estimates of state’s “goodness” to sophisticated measure based on the probability of a state leading to a goal
Heuristic Search and Expert Systems

Games and Heuristics search

- The search spaces are large enough to require heuristic pruning
- Most games are complex enough to suggest a rich variety of heuristic evaluations for comparison and analysis
- Games generally do not involve complex representational issues.
- Because each node of the state space has a common representation, a single heuristic may be applied throughout the search space

Games and Expert Systems

- The insights gained from simple games generalize to problems such as those found in expert systems applications, planning, intelligent control, and machine learning
Admissibility, Monotonicity and Informedness

- Heuristics that finds the shortest path to a goal whenever it exists are said to be **admissible**
- Whether any better heuristics are available, in what sense is one heuristic “better” than another? This is the **informedness** of a heuristic
- When a state is discovered by using heuristic search, is there any guarantee that the same state won’t be found later in the search at a cheaper cost (with a shorter path from the start state)? This is the property of **monotonicity**
DEFINITION

ALGORITHM A, ADMISSION, ALGORITHM A*

Consider the evaluation function \( f(n) = g(n) + h(n) \), where

- \( n \) is any state encountered in the search.
- \( g(n) \) is the cost of \( n \) from the start state.
- \( h(n) \) is the heuristic estimate of the cost of going from \( n \) to a goal.

If this evaluation function is used with the best_first_search algorithm of Section 4.1, the result is called algorithm \( A \).

A search algorithm is admissible if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

If algorithm \( A \) is used with an evaluation function in which \( h(n) \) is less than or equal to the cost of the minimal path from \( n \) to the goal, the resulting search algorithm is called algorithm \( A^* \) (pronounced "A STAR").

It is now possible to state a property of \( A^* \) algorithms:

All \( A^* \) algorithms are admissible.
The admissibility of A* algorithm is a theorem

Theorem

Any A* algorithm, i.e., one that uses a heuristic $h(n)$ such that $h(n) \leq h^*(n)$ for all $n$, is guaranteed to find the minimum path from $n$ to the goal, if such a path exists.
DEFINITION

MONOTONICITY

A heuristic function $h$ is monotone if

1. For all states $n_i$ and $n_j$, where $n_j$ is a descendant of $n_i$,

   $$h(n_i) - h(n_j) \leq \text{cost}(n_i, n_j),$$

   where $\text{cost}(n_i, n_j)$ is the actual cost (in number of moves) of going from state $n_i$ to $n_j$.

2. The heuristic evaluation of the goal state is zero, or $h(\text{Goal}) = 0$. 
Monotone property

- The search space is everywhere locally consistent with the heuristic employed
- The difference between the heuristic measure for a state and any one of its successors is bound by the actual cost of going between that state and its successor
- the heuristic is everywhere admissible, reaching each state along the shortest path from its ancestors
- the heuristic finds the shortest path to any state the first time that state is discovered, when a state is encountered a second time, it is not necessary to check whether the new path is shorter
Any monotonic heuristic is admissible

- Let $s_1, s_2, \ldots, s_g$ be the sequence of states, where $s_1$ is the start state and $s_g$ is the goal state

  $s_1$ to $s_2$: $h(s_1) - h(s_2) \leq \text{cost}(s_1, s_2)$ by monotone property

  $s_2$ to $s_3$: $h(s_2) - h(s_3) \leq \text{cost}(s_2, s_3)$ by monotone property

  $s_3$ to $s_4$: $h(s_3) - h(s_4) \leq \text{cost}(s_3, s_4)$ by monotone property

  \[ \vdots \]

  $s_{g-1}$ to $s_g$: $h(s_{g-1}) - h(s_g) \leq \text{cost}(s_{g-1}, s_g)$ by monotone property

  Summing each column and using the monotone property of $h(s_g) = 0$:

  path $s_1$ to $s_g$: $h(s_1) \leq \text{cost}(s_1, s_g)$

  This means that monotone heuristic $h$ is A* and admissible
Informedness

For two A* heuristics \( h_1 \) and \( h_2 \), if \( h_1(n) \leq h_2(n) \), for all states \( n \) and \( h_1(m) < h_2(m) \) in the search space, heuristic \( h_2 \) is said to be more informed than \( h_1 \).

- breadth-first search is equivalent to the A* algorithm with heuristic \( h \), such that \( h_1(x) = 0 \) for all states \( x \).
- This is, trivially, less than \( h^* \).
- \( h_2 \), the number of tiles out of place with respect to the goal state, is a lower bound for \( h^* \), i.e. \( h_1 \leq h_2 \leq h^* \)
- It follows that “the number of tiles out of place” heuristic is more informed than breadth-first search

In general,
- the more informed an A* algorithm, the less of the space it needs to expand to get the optimal solution
- the computations necessary to employ the more informed heuristic are not so inefficient as to offset the gains from reducing the number of states searched
Figure 4.12: Comparison of state space searched using heuristic search with space searched by breadth-first search. The portion of the graph searched heuristically is shaded. The optimal solution path is in bold. Heuristic used is $f(n) = g(n) + h(n)$ where $h(n)$ is tiles out of place.
Using Heuristics in Games

• Games have always been an important application area for heuristic algorithms

• Two-person games are more complicated than simple puzzles because of the existence of a “hostile” and essentially unpredictable opponent

• consider games whose state space is small enough to be exhaustively searched; here the problem is systematically searching the space of possible moves and countermoves by the opponent

• look at games in which it is either impossible or undesirable to exhaustively search the game graph

• Because only a portion of the state space can be generated and searched, the game player must use heuristics to guide play along a path to a winning state
nim game

• State space may be exhaustively searched
• to play this game, a number of tokens are placed on a table between the two opponents
• at each move, the player must divide a pile of tokens into two nonempty piles of different sizes
• for example, 6 tokens may be divided into piles of 5 and 1 or 4 and 2, but not 3 and 3
• the first player who can no longer make a move loses the game
Figure 4.13: State space for a variant of nim. Each state partitions the seven matches into one or more piles.
Minmax

- The opponents in a game are referred to as MIN and MAX
- MAX represents the player trying to win or MAXimize her advantage
- MIN is the opponent who attempts to MINimize MAX’s score
- MIN uses the same information and always attempts to move to a state that is worst for MAX
- MIN is allowed to move first
- Each leaf node is given a value of 1 or 0 depending on whether it is a win for MAX or for MIN
- Minmax propagates these values up the graph through successive parent nodes according to the rule:
  If the parent state is MAX node, give it the maximum value among its children
  If the parent is MIN node, give it minimum value of its children
- the value that is thus assigned to each state indicates the value of the best state that this player can hope achieve
Figure 4.14: Exhaustive minimax for the game of nim. Bold lines indicate forced win for MAX. Each node is marked with its derived value (0 or 1) under minimax.
Minimaxing to Fixed Ply Depth

- In applying minimax to more complicated games, it is seldom possible to expand the state space graph out to the leaf nodes.
- Instead, the state space is searched to a predefined number of levels, as determined by available resources of time and memory.
- This strategy is called an **n-ply look-ahead**, where *n* is the number of levels explored.
- As the leaves of this subgraph are not final states of the game, it is not possible to give them values that reflect a win or a loss.
- Instead, each node is given a value according to some heuristic evaluation function.
- The value that is propagated back to the root node is not an indication of whether or not a win can be achieved, but simply the heuristic value of the best state that can be reached in *n* moves from the start node.
• Look-ahead increases the power of a heuristic by allowing it to be applied over a greater area of the state space
• Minimax consolidates these separate evaluations into a single value of an ancestor state
• a simple might take the difference in the number of pieces belonging to MAX and MIN and try to maximize the difference between these piece measures
Figure 4.15: Minimax to a hypothetical state space. Leaf states show heuristic values; internal states show backed-up values.
Figure 4.16:
Heuristic measuring conflict applied to states of tic-tac-toe.

```
X
O
```

X has 6 possible win paths:

```
X

O
```

O has 5 possible wins:

```
X

O
```

\[ E(n) = 6 - 5 = 1 \]

```
X

O
```

X has 4 possible win paths;
O has 6 possible wins

\[ E(n) = 4 - 6 = -2 \]

```
X

O
```

X has 5 possible win paths;
O has 4 possible wins

\[ E(n) = 5 - 4 = 1 \]

Heuristic is \( E(n) = M(n) - O(n) \)

where \( M(n) \) is the total of My possible winning lines

\( O(n) \) is total of Opponent's possible winning lines

\( E(n) \) is the total Evaluation for state \( n \)