Heuristic Search

*Heuristic search* prefers to visit states that appear to be better.

*A* search visits states based on cost from initial to a given state plus *heuristic function*.

A *heuristic function* estimates the cost from a given state to the closest goal state.

```plaintext
function A*(initial, Expand, Goal,
            Cost, Heuristic)
  q ← New-Priority-Queue()
  Insert(initial, q, Heuristic(initial))
  while q is not empty
    do current ← Extract-Min(q)
    if Goal(current) then return solution
    for each next in Expand(current)
      do Insert(next, q, Cost(next) + Heuristic(next))
  return failure
```
IDA*: Iterative Deepening A* Search

function IDA*(\texttt{initial, EXPAND, GOAL, COST, HEURISTIC})

\begin{align*}
\text{limit} &\leftarrow \text{HEURISTIC}(\text{initial}) \\
\text{loop} & \quad \text{do result, limit} \leftarrow \text{Contour} \left( \text{initial, limit} \right) \\
& \quad \quad \text{if result then return result} \\
& \quad \quad \text{if limit} = \infty \text{ then return failure} \\
\end{align*}

function Contour(\texttt{current, limit})

\begin{align*}
\text{cost} &\leftarrow \text{Cost}(\text{current}) + \text{Heuristic}(\text{current}) \\
\text{if limit} < \text{cost} & \quad \text{then return null, cost} \\
\text{if Goal}(\text{current}) & \quad \text{then return solution, cost} \\
\text{new-limit} &\leftarrow \infty \\
\text{for each next in EXPAND(\text{current})} & \quad \text{do result, cost} \leftarrow \text{Contour}(\text{next, limit}) \\
& \quad \quad \text{if result then return solution, cost} \\
& \quad \quad \text{new-limit} \leftarrow \min(\text{new-limit, cost}) \\
\text{return} & \quad \text{failure, new-limit}
\end{align*}
Properties of A* Search

Let $n$ be a state/node.
Let $g(n)$ be the cost from the initial state to $n$.
Let $h(n)$ be the estimate from $n$ to a goal state.
Let $f(n) = g(n) + h(n)$.

$h$ is admissible if it is never an overestimate. 
*If $h$ is admissible, then $A^*$ finds optimal path.*
Proof Sketch: Let $f^*$ be optimal path cost.
Because $h$ never overestimates, then all states $n$ on optimal path have $f(n) \leq f^*$.

Because of priority queue, $A^*$ will visit optimal path before any suboptimal goal state.

Assume tree-structured state space ($b =$ branching factor, $d =$ goal depth), single goal state, each edge costs 1, and maximum error of $\epsilon$.

$A^*$ and IDA* visit $O(db^{\epsilon/2})$ states.
$A^*$ uses $O(db^{\epsilon/2})$ memory. IDA* uses $O(db)$. 
Performance of Heuristic Functions

Consider these 8-puzzle heuristic functions:

\( h_1 \): number of tiles in goal position.
\( h_2 \): Manhattan distance from tiles to goals.
Both never overestimate and \( h_1 \leq h_2 \)

Characterize by effective branching factor. If \( N \) states visited and solution depth \( d \), then solve for \( N = \sum_{i=0}^{d} x^i \)

---

Book Experiment Avoiding Bidirectional Edges

<table>
<thead>
<tr>
<th>( d )</th>
<th>States Visited (Effective BF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>IDS 52 (2.35) ( h_1 ) 10 (1.35) ( h_2 ) 7 (1.17)</td>
</tr>
<tr>
<td>8</td>
<td>IDS 569 (2.03) ( h_1 ) 42 (1.36) ( h_2 ) 14 (1.11)</td>
</tr>
<tr>
<td>12</td>
<td>IDS 5357 (1.92) ( h_1 ) 315 (1.47) ( h_2 ) 45 (1.19)</td>
</tr>
<tr>
<td>16</td>
<td>IDS 47271 (1.87) ( h_1 ) 2410 (1.52) ( h_2 ) 226 (1.28)</td>
</tr>
<tr>
<td>20</td>
<td>IDS 17646 (1.55) ( h_1 ) 764 (1.29) ( h_2 )</td>
</tr>
</tbody>
</table>
Local Search

A *local search* algorithm keeps track of one state at a time, using an evaluation function and a selection procedure to decide what state to visit next.

Local search gives up optimality guarantees in hopes of finding good solutions more efficiently. The main difficulty is local minima/maxima.
Local Search Algorithms

**function** `LOCAL-SEARCH(initial, Expand, Goal, Select)`

```plaintext
current ← initial
loop
  do if `Goal(current)` then return solution
  current ← `Select(Expand(current))`
```

Hill-Climbing, Gradient Descent:
Select state improving an evaluation function.

Random-restart hill-climbing:
Repeat hill climbing from random initial states.

Simulated Annealing:
Hill-climbing with randomized selection.

Genetic Algorithms:
Maintain a set of “current states.” Crossover generates new states from pairs of states.

Tabu Search: Like hill-climbing, but avoid recently visited states or recently used operators.