INFORMED SEARCH ALGORITHMS

CHAPTER 4, SECTIONS 1–2, 4
Outline

◊ Best-first search
◊ A* search
◊ Heuristics
function Tree-Search(problem, fringe) returns a solution, or failure

fringe ← Insert(Make-Node(Initial-State[problem]), fringe)

loop do
  if fringe is empty then return failure
  node ← Remove-Front(fringe)
  if Goal-Test[problem] applied to State(node) succeeds return node
  fringe ← InsertAll(Expand(node, problem), fringe)

A strategy is defined by picking the order of node expansion.

Tree search is called GENERAL-SEARCH of AIMA’s first edition (AIMA1ed), but the idea is always the same, whatever function names etc.
In uninformed search, we don’t try to evaluate which of the nodes on the frontier are most promising. We never look-ahead to the goal.

E.g., in uniform cost search we always expand the cheapest path. We don’t consider the cost of getting to the goal.

Often we have some other knowledge about the merit of nodes, e.g., going the wrong direction in Romania.
Heuristic-based Informed Searches

Merit of a frontier node:

◊ If we are concerned about the cost of the solution, we might want a notion of merit of how costly it is to get to the goal from that search node.

◊ If we are concerned about minimizing computation in search we might want a notion of ease in finding the goal from that search node.

◊ We will focus on the cost of solution notion of merit.
Heuristics

◊ The idea is to develop a domain specific heuristic function \( h(n) \).

◊ \( h(n) \) guesses the **cost of getting to the goal from node** \( n \).

◊ Heuristics are domain specific.
Best-first search algorithms

Best-first search is the generic name for the family of search algorithms that expand the most desirable node from the fringe first:

◊ insert in order of desirability and remove front

◊ insert to the end of the queue, and find most desirable while picking the front node from the fringe

These are equivalent. Remember that we also made the same comment talking about uninformed searches. They could be seen as using a Queue, a Priority Queue or Stack or just using different insertion/removal strategy to/from the queue that implemented the fringe.
Best-first search algorithms

Expand most desirable unexpanded node n using its merit - different algorithms will differ in how they measure merit.

◊ Greedy search

◊ A* search
Romania with step costs in km

Distances and the Straight-line distances as shown:

<table>
<thead>
<tr>
<th></th>
<th>Bucharest</th>
<th>Giurgiu</th>
<th>Urziceni</th>
<th>Hirsova</th>
<th>Neamt</th>
<th>Oradea</th>
<th>Zerind</th>
<th>Arad</th>
<th>Timisoara</th>
<th>Lugoj</th>
<th>Mehadia</th>
<th>Dobreta</th>
<th>Craiova</th>
<th>Sibiu</th>
<th>Fagaras</th>
<th>Rimnicu Vilcea</th>
<th>Pitesti</th>
<th>Vaslui</th>
<th>Iasi</th>
<th>Rimnicu Vilcea</th>
<th>Sibiu</th>
<th>Timisoara</th>
<th>Vaslui</th>
<th>Iasi</th>
<th>Zerind</th>
<th>Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>71</td>
<td>151</td>
<td>161</td>
<td>77</td>
<td>151</td>
<td>291</td>
<td>365</td>
<td>241</td>
<td>366</td>
<td>329</td>
<td>99</td>
<td>80</td>
<td>146</td>
<td>142</td>
<td>85</td>
<td>90</td>
<td>92</td>
<td>87</td>
<td>98</td>
<td>193</td>
<td>253</td>
<td>329</td>
<td>199</td>
<td>80</td>
<td>374</td>
</tr>
<tr>
<td>Urziceni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hirsova</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neamt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oradea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zerind</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timisoara</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lugoj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mehadia</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dobreta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Craiova</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sibiu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fagaras</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pitesti</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vaslui</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iasi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sibiu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timisoara</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vaslui</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iasi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zerind</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 4, Sections 1–2, 4
Greedy search

Evaluation function $f(n) = h(n)$

Here we use the $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest

Greedy search expands the node that appears to be closest to goal without thinking about the future: ”do what brings you the biggest reward now.”
Greedy search example
(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Fagaras
Properties of greedy search

Complete??
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g., Problem is getting from Iasi to Oradea.

Iasi → Neamţ → Iasi → Neamţ →

Complete in finite space with repeated-state checking

**Time??**
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??**
Properties of greedy search

Complete?? No—can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal??
Properties of greedy search

Complete?? No—can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No
A* search

Idea: avoid expanding paths that are already expensive by combining UniformCost (which was Optimal and Complete) and Greedy (which tends to be efficient).

Evaluation function \( f(n) = g(n) + h(n) \):

\( g(n) = \text{cost to reach } n \)
\( h(n) = \text{estimated cost to goal from } n \)
\( f(n) = \text{estimated total cost of path through } n \text{ to goal} \)
A* search example
(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Rimnicu Vilcea

(e) After expanding Fagaras
A* search and Admissible heuristics

Theorem: A* using TREE search and an admissible heuristic is optimal.

$h(n)$ is admissible if for all $n$, $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from $n$.

E.g., $h_{SLD}(n)$ never overestimates the actual road distance

Note: We also require $h(n) \geq 0$, so $h(G) = 0$ for any goal $G$. 
Optimality of A* using TREE Search

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G$.

![Diagram of a tree search problem](image)

We will show that even though $G_2$ is in the queue, it will not be picked (and lead to suboptimal solution) before $n$. 
Optimality of A* using TREE Search

\[ f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0 \]
\[ > g(G) \quad \text{since } G_2 \text{ is suboptimal} \]

Also,
\[ f(n) = g(n) + h(n) < g(G) \quad \text{since } h \text{ is admissible} \]

Hence, \( f(G_2) > f(n) \), and A* will never select \( G_2 \) for expansion.
Optimality of A*

In contrast to tree search, graph search may discard the optimal path to a repeated state if it is not the first one generated, so the optimality no longer holds.

◊ Solution 1: Discard the more expensive of any two paths found to the same node (extra bookkeeping).

◊ Solution 2: Use a consistent heuristic
A heuristic is *consistent* if

\[ h(n) \leq c(n, a, n') + h(n') \]
A heuristic is *consistent* if

\[ h(n) \leq c(n, a, n') + h(n') \]

If \( h \) is consistent, we have

\[
\begin{align*}
f(n') & = g(n') + h(n') \\
& = g(n) + c(n, a, n') + h(n') \\
& \geq g(n) + h(n) \\
& \geq f(n)
\end{align*}
\]

◊ For consistent (also called monotonic) heuristics, the values of \( f(n) \) along any path are non-decreasing.

◊ Consistency is a stricter requirement than admissibility:
all consistent heuristics are admissible, but not necessarily the other way around (although most admissible heuristics are also consistent).
Optimality of A* Using GRAPH Search

**Lemma:** A* expands nodes in order of increasing \( f \) value*

Gradually adds “\( f \)-contours” of nodes
Contour \( i \) has all nodes with \( f < f_i \) and \( f_i < f_{i+1} \)
A* is optimal, since it cannot expand $f_{i+1}$ until $f_i$ is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

Note: Breadth-first adds layers, uniform cost search adds concentric bands...
Properties of $A^*$

Complete??
Properties of $A^*$

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time??**
Properties of A*

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time??** Exponential in [relative error in $h \times$ length of soln.]

**Space??**
Properties of A*

**Complete??** Yes, unless there are infinitely many nodes with \( f \leq f(G) \)

**Time??** Exponential in [relative error in \( h \times \) length of soln.]

**Space??** Keeps all nodes in memory

**Optimal??**
Properties of $A^*$

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time??** Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

**Space??** Keeps all nodes in memory

**Optimal??** Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

$A^*$ expands all nodes with $f(n) < C^*$

$A^*$ expands some nodes with $f(n) = C^*$

$A^*$ expands no nodes with $f(n) > C^*$
Properties of A*

A* is optimally efficient: among the search algorithms that start from the root and search using the same heuristic, it is the most efficient (may consider extra only those nodes with f cost exactly equal to C*).

Space is the main problem with A* like other Graph Search algorithms that keeps track of a Closed/Explored list.

Note: The graph search algorithms typically need to check whether the reached state is a repeated state, but Version 3 of AIMA allows for checking repeated nodes in the given pseudocode, allowing different algorithms to implement what they need (to check for states or nodes). In any case, space requirements are often large.
Iterative deepening A*: similar to IDDS (Iterative Deepening DFS), this one increases the limit of f-costs at each iteration.

Memory-bounded A*: expands the best leaf until memory is full; then drops the worst leaf node, backs up the value of f-cost of the forgotten node to its parent (to take it up if nothing else is better).
Admissible Heuristics

So far we have seen one heuristic - how do we devise new ones?

Also how important is it to find a good heuristic?
Heuristics - What for?

E.g., for the 8-puzzle:

![Start State](image)

![Goal State](image)

What are is the branching factor?
8-puzzle

- Average solution takes 20 steps
- $b$ is about 3 (exhaustive search would look at $b^{20}$ roughly $10^{10}$ states)
- there are only 9! different states ($= 350,000$)

Hence, a good heuristic can reduce the search space drastically.
Admissible heuristics

E.g., for the 8-puzzle:

\begin{align*}
  h_1(n) &= \text{number of misplaced tiles} \\
  h_2(n) &= \text{total Manhattan distance} \\
          &\quad\text{(i.e., no. of squares from desired location of each tile)}
\end{align*}

\begin{align*}
  h_1(S) &= ?? \\
  h_2(S) &= ??
\end{align*}
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]
\[ (\text{i.e., no. of squares from desired location of each tile}) \]

\[ h_1(S) = ?? \quad 7 \]
\[ h_2(S) = ?? \quad 4+2+2+2+3+3 = 18 \quad (\text{tile 5 would need to move 4 steps, tiles 3 and 2 need to move 3 steps each}) \]
Which heuristic is better

Remember:

◊  We want admissible heuristics which always underestimate the remaining distance to the goal.

◊  We don’t want the trivially admissible heuristic which says the remaining distance to the goal is 0 (or the like)
Dominance

If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$ and is better for search
Dominance

Typical search costs:

\[ d = 14 \quad \text{IDS} = 3,473,941 \text{ nodes} \]
\[ A^*(h_1) = 539 \text{ nodes} \]
\[ A^*(h_2) = 113 \text{ nodes} \]

\[ d = 24 \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \]
\[ A^*(h_1) = 39,135 \text{ nodes} \]
\[ A^*(h_2) = 1,641 \text{ nodes} \]

Note how much improvement there is between A* and IDS, but also between A* with a good and average heuristic (factor of 10-30).
How to Compare Heuristics

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
<td>1.45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
<td>1.22</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
<td>1.23</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>

N is found by averaging over 100 instances of the 8-puzzle problem, with solution lengths of 2,4,...20. I.e. given $d$, we generate 100 8-puzzle problems that has a solution at depth $d$ (since the search may be slightly different for a given starting state, we
average over different starting states).
Effective Branching Factor

How many nodes are examined by a heuristic depends on the depth of the solution and the maximum depth of that particular problem, ordering of the branches etc...

One good quantitative measure to compare two heuristics is to look at their effective branching factors: the branching factor of a *uniform tree* of depth $d$ in order to contain $n$ nodes
Effective Branching Factor

Effective Branching Factor $b$ can be found by solving for $b$ in the following formula (we know $N$ (number of nodes visited) and $d$ (depth of solution)), assuming a uniform tree.

Remember: $N = 1 + b + b^2 + \ldots + b^d = \frac{b^{(d+1)} - 1}{b - 1}$

The idea is to judge how the heuristics is able to focus the search (can they zoom in to the goal or are they almost blindly searching?).
### Dominance and Effective Branching Factor

<table>
<thead>
<tr>
<th>d</th>
<th>IDS</th>
<th>A*(h₁)</th>
<th>A*(h₂)</th>
<th>IDS</th>
<th>A*(h₁)</th>
<th>A*(h₂)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
<td>1.45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
<td>1.22</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
<td>1.23</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>

N is found by averaging over 100 instances of the 8-puzzle problem, with solution lengths of 2,4,...20.

I.e. given d, we generate 100 8-puzzle problems that have a solution at depth d (since the search may be slightly different
for a given starting state, we average over different starting states).
Generating Admissible Heuristics

A version of a problem with fewer constraints or fewer restrictions on the actions is called a *relaxed* version of the problem.

Admissible heuristics can be generated using relaxed problems.
Generating Admissable Heuristics

Relaxed problems (hence heuristics) can be automatically generated if the problem can be defined using a formal language.

The relaxed form of the operators of the 8-puzzle can be written as:

A tile can move from square A to B, if A is adjacent to B and B is blank.

◇ A tile can move from square A to B, if A is adjacent to B
◇ A tile can move from square A to B, if B is blank
◇ A tile can move from square A to B

*Which one corresponds to h1 or h2?*
Relaxed problems

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution (A tile can move from square A to B)

If the rules are relaxed so that a tile can move to any adjacent square even if occupied, then $h_2(n)$ gives the shortest solution (A tile can move from square A to B, if A is adjacent to B)

What about: "A tile can move from square A to B, if B is blank"? (exercise, close to $h_1$).

Key point: The cost of an optimal solution to a relaxed problem can be used as an admissible heuristic for the original problem! Why?

Key point 1: the optimal solution cost of a relaxed problem
is no greater than the optimal solution cost of the real problem - hence the generated heuristic is admissible.

Key point 2: Finding the cost of the optimal solution to the relaxed problem should be done without searching.
Generating Admissible Heuristics

ABSOLVER program can automatically generate heuristics using the "relaxed problem" method (Prieditis 1993)

It generated the best heuristic thus far, for 8-puzzle and the first useful heuristic to the Rubik’s cube.
Generating Admissible Heuristics

Another way to generate heuristics is to use solutions to a sub-problem of the given problem.

e.g. only put the first 4 tiles into their correct positions.

It turns out that this can be substantially better than the relaxed problem solutions.
Heuristics Issues

◊ Can we combine heuristics to find a better one?
◊ Learning
◊ What about the computational cost of the heuristic?!
Heuristics Issues

◊ Can we combine heuristics to find a better one?

Use \( h(n) = \max(h_1(n), \ldots, h_m(n)) \)

If the component heuristics are admissible, \( h \) will also be admissible.

◊ Learning

You can learn to "correct the heuristic" used.

◊ What about the computational cost of the heuristic?!

It is clear that one cannot do a breadth-first-search to compute a heuristic value!

◊ Features
Use weighted combinations of different features
e.g. for chess, number of pieces left, the area you control on the board...