Informed Search and Heuristic Functions

- For informed search, we use *problem-specific* knowledge to guide the search.

Topics:

- Best-first search
- A* search
- Heuristics
Recall: General Tree Search

```
function Tree-Search(problem) returns a solution or failure
initialize the search tree by the initial state of problem
loop do {
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion (according to some strategy)
    - remove the leaf node from the frontier
    if the node satisfies the goal state then return the solution
    expand the node and add the resulting nodes to the search tree
}
```
Informed (Heuristic) Search

• **Idea**: use an *evaluation function* for each node
  - estimate of “desirability” or proximity to a goal.
• Expand the most desirable unexpanded node

The evaluation function is defined as:

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

- \( g(n) \) = cost from root to node \( n \)
- \( h(n) \) = estimated cost from node \( n \) to the goal
  - heuristic function

In uniform-cost search:

\[ f(n) = g(n) \]
Informed (Heuristic) Search

- **Idea**: use an *evaluation function* for each node
  - estimate of “desirability” or proximity to a goal.
- Expand the most desirable unexpanded node
- Most generally we have:
  - Evaluation function: \( f(n) = g(n) + h(n) \)
    - \( g(n) \) = cost from root to node \( n \)
    - \( h(n) \) = estimated cost from node \( n \) to the goal
      - \( h(n) \) – *heuristic function*
    - \( f(n) \) = estimated total cost of path through \( n \) to goal
  - Thus for uniform-cost search \( f(n) = g(n) \).
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$
  
  \[ h(n) = \text{estimate of cost from } n \text{ to the closest goal} \]

- So, $g(n) = 0$
  
  - i.e. the cost from the root to $n$ is not considered.

- E.g., $h_{\text{SLD}}(n) = \text{straight-line distance from } n \text{ to Bucharest}$

- Greedy search expands the node that \textit{appears} to be closest to goal
Example: Romania with step costs in km

<table>
<thead>
<tr>
<th>City</th>
<th>Bucharest</th>
<th>Giurgiu</th>
<th>Urziceni</th>
<th>Hirsova</th>
<th>Eforie</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>Pitesti</th>
<th>Oreadea</th>
<th>Neamt</th>
<th>Sibiu</th>
<th>Timisoara</th>
<th>Vaslui</th>
<th>Pitesti</th>
<th>Rimnicu Vilcea</th>
<th>Iasi</th>
<th>Giurgiu</th>
<th>Eforie</th>
<th>Zerind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucharest</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>Giurgiu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>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>
<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>
<td></td>
<td></td>
</tr>
<tr>
<td>Eforie</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>242</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>71</td>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>118</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>80</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>80</td>
<td>99</td>
<td>211</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>380</td>
<td>92</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>92</td>
<td>92</td>
<td>142</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>Veluie</td>
<td>193</td>
<td>98</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>71</td>
<td>140</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>118</td>
<td>75</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>87</td>
<td>99</td>
<td>138</td>
<td>101</td>
<td>211</td>
<td>92</td>
<td>92</td>
<td>85</td>
<td>90</td>
<td>86</td>
<td>142</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td></td>
<td></td>
<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>

Straight-line distance to Bucharest:

- Arad: 366 km
- Bucharest: 0 km
- Craiova: 160 km
- Dobreta: 242 km
- Eforie: 161 km
- Fagaras: 178 km
- Giurgiu: 77 km
- Hirsova: 151 km
- Iasi: 226 km
- Lugoj: 244 km
- Mehadia: 241 km
- Neamt: 234 km
- Oradea: 380 km
- Pitesti: 98 km
- Rimnicu Vilcea: 193 km
- Sibiu: 253 km
- Timisoara: 329 km
- Urziceni: 80 km
- Vaslui: 199 km
- Zerind: 374 km
Greedy search example

Arad
366
Greedy search example

Sibiu 253

Timisoara 329

Zerind 374

Arad
Greedy search example
Greedy search example
Other examples

- Games (i.e. as a search technique in *adversarial search*)
- Others?
Properties of greedy search

Complete: ??
Properties of greedy search

**Complete:** No – can get stuck in loops,

- E.g., with Oradea as goal,
  \[\text{Iasi} \rightarrow \text{Neamt} \rightarrow \text{Iasi} \rightarrow \text{Neamt} \rightarrow\]

  Complete in finite space with repeated-state checking

**Time:** ??
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:** ??
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
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

- Note that this is for an (offline) breadth-first tree-search version of the algorithm.
- An (online) depth-first agent could perform in constant space using via local search (later).

Optimal: ??
Properties of greedy search

Complete: No – can get stuck in loops,
  • E.g., Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ 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
  • Note that this is for an (offline) breadth-first tree-search version of the algorithm.
  • An (online) depth-first agent could perform in constant space using via local search (later).

Optimal: No
A* search

Idea:

• Try to avoid expanding paths that look to be expensive
  • Evaluation function $f(n) = g(n) + h(n)$
  • $g(n) =$ cost so far to reach $n$
  • $h(n) =$ estimated cost to the goal from $n$
  • $f(n) =$ estimated total cost of path through $n$ to goal

• Expand the node where the cost so far, plus the estimated cost, is minimal.

• Note that $f(n)$ is a heuristic function. It may not give the best value.

• A good choice of a heuristic function is crucial for good performance.
A* search

A* search (ideally) uses an *admissible* heuristic

- Let $h^*(n)$ be the *true* (unknown) cost from $n$ to the goal.
- A heuristic function $h(n)$ is admissible just if:
  $h(n) \leq h^*(n)$
  
  So $h(n)$ never overestimates the cost.

- Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal $G$.

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

*Theorem*: A* search is optimal

*Corollary*: Uniform cost search is optimal (why?)
A* search example

Arad
366 = 0 + 366
A* search example

- **Zerind**
  - 393 = 140 + 253

- **Arad**
  - 447 = 118 + 329

- **Sibiu**
  - 393 = 140 + 253

- **Timisoara**
  - 447 = 118 + 329

- **Zerind**
  - 449 = 75 + 374
A* search example
A* search example
A* search example
A* search example
Optimality of $A^*$ (standard proof)

- Suppose $G_2$ is a suboptimal goal.
- Let $n$ be an unexpanded node on a shortest path to an optimal goal $G$:

![Diagram](image)

- Then:
  \[
  f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0
  \]
  \[
  > g(G) \quad \text{since } G_2 \text{ is suboptimal}
  \]
  \[
  \geq f(n) \quad \text{since } h \text{ is admissible}
  \]
- Since $f(G_2) > f(n)$, $A^*$ will never select $G_2$ for expansion
Optimality of A* (another view)

- **Lemma**: A* expands nodes in order of increasing $f$ value.
- Gradually adds “$f$-contours” of nodes
  - Cf.: breadth-first adds “layers”
- Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$
Properties of $A^*$

Complete: ??
Properties of A* 

Complete: Yes, unless there are \( \infty \) many nodes with \( f \leq f(G) \)

Time: ??
Properties of A*

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

**Time:** Exponential in \([\text{relative error in } h \times \text{length of soln.}]\)

**Space:** ??
Properties of A*

**Complete:** Yes, unless there are $\infty$ 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

$\Rightarrow$ So exponential

**Optimal:** ??
Properties of A*

Complete: Yes, unless there are ∞ 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

- A* expands all nodes with \( f(n) < C^* \), where \( C^* = \text{cost of optimal solution} \)
- A* expands some nodes with \( f(n) = C^* \)
- A* expands no nodes with \( f(n) > C^* \)
Admissible heuristics

For the 8-puzzle:

$h_1(n) =$ number of misplaced tiles

$h_2(n) =$ total Manhattan distance (i.e., number of squares from desired location of each tile)
Admissible heuristics

For the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
Admissible heuristics

For the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]
\[ \quad \text{(i.e., number of squares from desired location of each tile)} \]
Admissible heuristics

For the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., number of squares from desired location of each tile)

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array}
\quad \quad
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \\
\end{array}
\]

Start State   Goal State

\[ h_1(S) = ?? \]
\[ h_2(S) = ?? \]
Admissible heuristics

For the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., number of squares from desired location of each tile)

\[
\begin{bmatrix}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{bmatrix}
\]

Start State

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \\
\end{bmatrix}
\]

Goal State

\[ h_1(S) = 6 \]

\[ h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14 \]
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.

- Typical search costs for 8 puzzle:
  - \( d = 14 \) IDS = 3,473,941 nodes
    \[ A^*(h_1) = 539 \text{ nodes} \]
    \[ A^*(h_2) = 113 \text{ nodes} \]
  - \( d = 24 \) IDS \( \approx \) 54,000,000,000 nodes
    \[ A^*(h_1) = 39,135 \text{ nodes} \]
    \[ A^*(h_2) = 1,641 \text{ nodes} \]

- For any admissible heuristics \( h_a, h_b \),
  \[ h(n) = \max(h_a(n), h_b(n)) \]
  is also admissible and dominates \( h_a, h_b \).
Determining admissible heuristic functions

Relaxed problems:

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.
  - E.g.: If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Key point: The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.
Determining admissible heuristic functions

Relaxed problems:

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem
- E.g.:
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
Determining admissible heuristic functions

Relaxed problems:

- Admissible heuristics can be derived from the \textit{exact} solution cost of a \textit{relaxed} version of the problem
- E.g.:
  - If the rules of the 8-puzzle are relaxed so that a tile can move \textit{anywhere}, then $h_1(n)$ gives the shortest solution
  - If the rules are relaxed so that a tile can move to \textit{any adjacent square}, then $h_2(n)$ gives the shortest solution

Key point:
The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem
Relaxed problems contd.

- Well-known example: *travelling salesman problem* (TSP)
- Find the shortest tour visiting all cities exactly once

![Graphs](image.png)

- *Minimum spanning tree* can be computed in $O(n^2)$ and is a lower bound on the shortest (open) tour
Summary: Heuristic functions

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest $h$
  - incomplete and not always optimal
- A* search expands lowest $g + h$
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems
Local Search: Outline

We consider next *local* search, where we maintain a single current state.

- Iterative improvement algorithms
- Hill-climbing
- Very briefly:
  - Simulated annealing
  - Local beam search
Iterative improvement algorithms

- Idea: In many optimization problems, the *path* to the goal is irrelevant.
  - The goal state itself is the solution
  - E.g. the *n*-queens problem
- So we may formulate a problem so that:
  state space = set of “complete” configurations
- Examples:
  - find *optimal* configuration, e.g., TSP
  - find configuration satisfying constraints, e.g., timetable
  - also, e.g. propositional satisfiability (SAT)
- In such cases, we can use *iterative improvement* algorithms
  - Keep a single “current” state; try to improve it
  - Uses constant space; suitable for online as well as offline search
Example: Travelling Salesperson Problem

- Start with any complete tour, perform pairwise exchanges

- Variants of this approach get within 1% of optimal very quickly with thousands of cities.
Example: *n*-queens

- Goal: Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal.
Example: $n$-queens

- Goal: Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
- Move a queen to reduce number of conflicts.

Almost always solves $n$-queens problems almost instantaneously for very large $n$, e.g., $n = 1,000,000$
Hill-climbing (or gradient ascent/descent)

- Idea: Take the best move from a given position
- Aka *greedy local search*.
- “Like climbing a mountain in thick fog with amnesia”
Hill-climbing

Function Hill-Climbing(problem) returns a state that is a local maximum

inputs: problem a problem

local variables: current a node
neighbor a node

current ← Make-Node(Initial-State[problem])

loop do
    neighbor ← a highest-valued successor of current
    if Value[neighbor] ≤ Value[current] then return State[current]
    current ← neighbor
end
Hill-climbing contd.

Useful to consider *state-space landscape*

![Graph showing state-space landscape with terms: objective function, global maximum, local maximum, "flat" local maximum, shoulder, current state, state space.](image-url)
Hill-climbing contd.

• Hill climbing often gets stuck:

  Local Maxima: I.e. local “peaks”.
    E.g. 8-queens gets stuck 86% of the time.
  Ridges: Essentially give a series of local maxima.
    Difficult for hill-climbing to navigate
  Plateaux: A plateau is a flat area in the search space.
    Search degenerates to exhaustive search, or may loop.
Hill-climbing: Strategies if stuck

- **Random-restart hill climbing**: Overcomes local maxima
  - Trivially complete *if* a goal is known to exist.
- **Random sideways moves**: Escape from shoulders but may loop on flat maxima
  - Can also define a hill-climbing version of depth-first search.
    (But then no longer a *local* search.)
Another Example: Propositional Satisfiability

- Goal: Find a *satisfying assignment* for a set of clauses in CNF.
- E.g.

\[(p \lor q \lor \neg r) \land (\neg p \lor r) \land (\neg p \lor \neg q)\]

is satisfied by setting: \(p = true, q = false, r = true\).
Propositional Satisfiability

- Outline of an algorithm:

  Function `Sat(problem)` returns a solution or failure
  Assign truth values arbitrarily to the set of propositional variables
  loop do {
    if the truth assignment satisfies `problem`
      then return the assignment
    if timeout then return failure
    Find `l` such that \( \overline{\overline{l}} \) gives the largest increase in clauses satisfied
    Change the truth value of `l` to \( \overline{\overline{l}} \).
  }

  If `l` is `p` then \( \overline{\overline{l}} \) is \( \neg p \);
  if `l` is \( \neg p \) then \( \overline{\overline{l}} \) is `p`.
Propositional Satisfiability

- This algorithm, when proposed in the 1990's, worked very well.
- The algorithm also featured random restarts. (i.e. after a while reassign all variable and start over).
  - It handily beat all previous algorithms (notably DPLL).
- Subsequent work in satisfiability has led to huge improvements over the naive greedy algorithm.
- Aside: Another thing that this work pointed out was the importance of choice of test instances.
  - DPLL (and other algorithms) appeared to work well because it turned out they were often tested on easy instances.
Simulated annealing

- **Goal:** Avoid local maxima
  - Local maxima is the biggest problem with local search.
- **Idea:** Take a step in a direction other than the best, from time to time.
  - Try to escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency
  - These steps are designed to get the solver out of a possible local maximum
- **The step size varies.**
  - As time passes the step size and probability of a non-best step decreases.
- **Simulated annealing has proven very effective in a wide range of problems, including VLSI layout, airline scheduling, etc.**
Local beam search

Idea:

- Begin with $k$ randomly-generated states.
- Keep $k$ states instead of 1; choose top $k$ of all their successors
- Not the same as $k$ searches run in parallel!
- Searches that find good states recruit other searches to join them

Problem:
Quite often, all $k$ states end up on same local hill

Variant: Stochastic beam search:
Choose $k$ successors randomly, biased towards good ones
- Observe the analogy to natural selection!