Search
Search Basics

- **Search** is finding a sequence of actions that achieve a goal from an initial state.

- Idea: Find sequence of actions before doing any actions.

- Idea: An agent that can predict the results of its actions can choose better actions.

- Assumptions: Actions are deterministic. All relevant features of states can be perceived. Can tell whether a state satisfies the goal.
A state-space problem consists of

- a set of *states*
- a subset of states called the *start states*
- a set of *actions*
- an *action function* that maps from a state and an action to a state
- a set of *goal states*, or, equivalently, a Boolean function, \( \text{goal}(s) \), that is true when \( s \) is a goal state
- a function that measures the quality of a solution
Typically, state-space problems are transformed to searching a directed graph

- A DG is a set $N$ of nodes/vertices and a set $A$ of arcs/edges.
- States are mapped to nodes, actions to arcs.
- Typically, the DG is too large to be explicitly stored. Instead, the action function is used to generate arcs as needed.
Procedure `Search(G, S, goal)`

Inputs: $G$: graph with nodes $N$ and arcs $A$

$S$: set of start nodes

`goal`: Boolean function of nodes

Output: path from $s$ to $g$ s.t. $s \in S$ and $goal(g)$
or null if no solution paths are found

`Frontier` ← $\{(s) | s \in S\}$  // a set of paths

while `Frontier` is not empty

remove a path $p = (s, \ldots, t)$ from `Frontier`

if $goal(t)$ then return $p$

for each arc from $t$ to $n$

insert $(s, \ldots, t, n)$ into `Frontier`

return null
### Example Search Problems

<table>
<thead>
<tr>
<th>Uninformed Search</th>
<th>Heuristic Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-puzzle, 15-puzzle</td>
<td>Tower of Hanoi</td>
</tr>
<tr>
<td>$n$-queens</td>
<td>Monkey and bananas</td>
</tr>
<tr>
<td>Cryptarithmetic (TWO + TWO = FOUR)</td>
<td>Airline booking</td>
</tr>
<tr>
<td>Missionaries and cannibals</td>
<td>Traveling salesman</td>
</tr>
<tr>
<td>Rubik's cube</td>
<td>Device assembly</td>
</tr>
<tr>
<td>Blocks-world</td>
<td>Calculus, algebra problems</td>
</tr>
</tbody>
</table>
Uninformed Search

- Depth-First Search: Maintain frontier as a **stack** (last-in, first-out). Prefers to remove longer paths from frontier. DFS can be improved by:
  - Cycle checking: Don’t put paths with cycles on the frontier.

- Breadth-First Search: Maintain frontier as a **queue** (first-in, first-out). Prefers to remove shorter paths from frontier. BFS can be improved by:
  - Multiple path pruning: There might be many paths that end with a node \( t \). Don’t put more than one on frontier.
Iterative Deepening

- BFS will find the shortest-path solution, but uses $O(b^d)$ time and space ($b = \text{branching factor}, d = \text{depth to solution}$).
- DFS uses $O(bd')$ space, but $d'$ might be the depth of a much longer solution.
- Iterative deepening uses DFS, but limits the depth of the search. Uses $O(bd)$ space (still $O(b^d)$ time).

\begin{verbatim}
bound ← 0

do
    solution ← DFS(G, S, goal, bound)
    bound ← bound + 1

while solution = null
\end{verbatim}
Heuristic Search

- Uninformed search algorithms do not consider whether a path appears to be close to the goal.
- Heuristic search prefers paths that appear better.
- A heuristic function (denoted $h$) estimates the cost from a given state to the goal. [For a path $p = (s, \ldots, t)$, use $h(p) = h(t)$.]
- Best-first search prefers paths $p$ with a lower value for $h(p)$.
- $A^*$ search prefers paths $p$ with lowest $f(p) = \text{cost}(p) + h(p)$. I.e., the cost of the path so far plus the estimated remaining cost to the goal.
A* Search Algorithm

- A* search is implemented by treating the frontier as a priority queue ordered by $f$.
- A* search finds the optimal path if $h$ never overestimates the remaining cost (called admissibility).
- If $h$ is admissible, A* search is $O(m)$ where $m$ is the number of paths with $f(p) \leq \text{cost of optimal path}$.
- [Not in book] The contour for cost $v$ consists of all states $n$ with $f(n) \leq v$. Typically, A* searches $f(n) \leq v$ before $f(n) > v$. 
Assume the state space is tree-structured with:
\[ b = \text{branching factor}, \]
\[ d = \text{depth of closest solution}, \]
\[ m = \text{maximum depth of state space}, \]
\[ l = \text{depth bound} \]
### ID Performance

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Paths Visited</th>
<th>Paths Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Depth-First</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>DFS (bounded)</td>
<td>$O(b^l)$</td>
<td>$O(bl)$</td>
</tr>
<tr>
<td>Iterative Deep.</td>
<td>$O(b^d)$</td>
<td>$O(bd)$</td>
</tr>
</tbody>
</table>

BFS and ID return optimal (shortest) solution.

Ratio of states visited is \[
\frac{\text{ID}}{\text{BFS}} \approx \frac{b}{b - 1}
\]
ID Performance

DFS with depth limit $l$ visits this many paths.

$$b^0 + b^1 + \ldots + b^l = \frac{b^{l+1} - 1}{b - 1} < \frac{b^{l+1}}{b - 1}$$

ID adds this for $l$ from 0 to $d$

$$\frac{b^1}{b - 1} + \frac{b^2}{b - 1} + \ldots + \frac{b^{d+1}}{b - 1}$$

This is equal to

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

which is $O(b^d)$
Notation for A* Search

- $n$: variable standing for a state
- $g(n)$: the cost from the initial state to $n$.
- $h(n)$: the estimate from $n$ to a goal state.
- $f(n) = g(n) + h(n)$. 
- Each action costs at least 1 unit.
- Number of actions are finite.
- $h$ is admissible ($h$ is never an overestimate).

Under above conditions, A* finds optimal path.
If above conditions, $h$ has at most $\epsilon$ error, and the search space is a uniform tree with one goal state, then A* searches at most $\epsilon/2$ from solution path.
Let $f^*$ be optimal path cost.

Because $h$ never overestimates, then all states $n$ on optimal path have $f(n) \leq f^*$.

Any nonoptimal goal state $n'$ has $f(n') > f^*$.

Because of priority queue, A* will visit states on optimal path before any nonoptimal goal state.

Other conditions prevent infinite search in a flat region of the state space.
Assume tree-structured state space ($b = \text{branching factor}, \ d = \text{goal depth}$), single goal state, each edge costs 1 and is reversible, and maximum error of $\epsilon$.

Any state $n$ more than $\epsilon/2$ off of solution path has $f(n) = g(n) + h(n) > f^*$.

All states $n$ on solution path have $f(n) = g(n) + h(n) \leq f^*$.

$A^*$ and IDA* visit $O(db^{\epsilon/2})$ states.

$A^*$ uses $O(db^{\epsilon/2})$ memory. IDA* uses $O(db)$. 
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

- Let $N$ states be visited and solution depth be $d$.
- Solve for $x$ in $N = \sum_{i=0}^{d} x^i$
## Experiment Avoiding Reverse Moves

<table>
<thead>
<tr>
<th>$d$</th>
<th>States Visited (Effective BF)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID</td>
</tr>
<tr>
<td>4</td>
<td>52 (2.35)</td>
</tr>
<tr>
<td>8</td>
<td>569 (2.03)</td>
</tr>
<tr>
<td>12</td>
<td>5357 (1.92)</td>
</tr>
<tr>
<td>16</td>
<td>47271 (1.87)</td>
</tr>
<tr>
<td>20</td>
<td>17646 (1.55)</td>
</tr>
</tbody>
</table>