INFORMED SEARCH ALGORITHMS

CMSC 421: Chapter 4, Sections 1–2
Motivation

♢ In Chapter 3 we were talked about trial-and-error search
♢ In the worst case, most searches take exponential time (unless P=NP)
♢ Can sometimes do much better on the average, using *heuristic* techniques

*Heuristic:*

♢ Rule of thumb, simplification, or educated guess
♢ Reduces the search for solutions in domains that are difficult and poorly understood.
♢ Depending on what heuristic you use, you won’t necessarily find an optimal solution, or even a solution at all.
Heuristic tree search

function Tree-Search(problem) returns a solution, or failure
    fringe ← a list containing Make-Node(Initial-State[problem])
    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)
    fringec = a list of the nodes that have been generated but not expanded:
Heuristic tree search

function Tree-Search(\textit{problem}) returns a solution, or failure

\textit{fringe} ← a list containing \textit{Make-Node}(\textit{Initial-State}[\textit{problem}])

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

Heuristic choice in search algorithms: \textbf{what node to expand next}

Use an \textit{evaluation function} \( f(n) \) for each node \( n \)

– estimate of "desirability"

⇒ Expand most desirable unexpanded node

\textbf{InsertAll} keeps \textit{fringe} sorted in decreasing order of desirability,

i.e., if \textit{fringe} = \( \langle s_1, s_2, \ldots, s_k \rangle \)

then \( f(s_1) \leq f(s_2) \leq \ldots \leq f(s_k) \)

Thus \textbf{Remove-Front} always gets a most-desirable node
Heuristic graph search

function Graph-Search(problem, fringe) returns a solution, or failure

closed ← an empty set
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, State[node]) then return node
    if State[node] is not in closed then
        add State[node] to closed
        fringe ← InsertAll(Expand(node, problem), fringe)
    end

Same as for Tree-Search:
    Use InsertAll to keep fringe sorted in decreasing order of desirability
Recall that we want to get from Arad to Bucharest:

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu V.</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy search

Heuristic function $h(n) = \text{estimate of cost from } n \text{ to the closest goal}$

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

Greedy search uses $f(n) = h(n)$,
i.e., keeps fringe ordered in increasing value of $h$

hence always expands whatever node appears to be closest to a goal
Greedy search example

straight-line distance to Bucharest
Greedy search example
Greedy search example
Greedy search example
Properties of greedy search

Complete?
Properties of greedy search

**Complete?** No. Can get stuck in loops:

Iasi \rightarrow Neamt \rightarrow Iasi \rightarrow Neamt \rightarrow

Complete in finite space with repeated-state checking

**Time?**
**Properties of greedy search**

**Complete?** No. Can get stuck in loops:

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:
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:

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

Problem with terminology:

*Greedy search* is not the same as an ordinary *greedy algorithm*.

An ordinary greedy algorithm doesn’t remember all of *fringe.*
It remembers only the current path, and never backtracks. Hence:

♦ Repeated-state checking cannot make it complete
♦ It runs in time $O(l)$ if it finds a solution of length $l$
A* tree search

Idea: avoid expanding paths that are already expensive

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

$g(n) =$ cost so far to reach $n$
$h(n) =$ estimated cost to goal from $n$
$f(n) =$ estimated total cost of path through $n$ to goal

Optimality requirement for A* tree search:

A* needs an **admissible** heuristic, i.e., $0 \leq h(n) \leq h^*(n)$

where $h^*(n)$ is the **true** cost from $n$.

(Thus $h(G) = 0$ for any goal $G$.)

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

**Theorem**: If the optimality requirement is satisfied, then A* tree search never returns a non-optimal solution
A* tree search

Completeness requirement for A* tree search:

No infinite path has a finite cost

Theorem: On any solvable problem that satisfies the completeness requirement, A* tree search returns a solution.

Corollary: If the optimality requirement also is satisfied, then A* tree search returns an optimal solution.
Romania with step costs in km

Recall that we want to get from Arad to Bucharest:

- Arad
- Bucharest
- Craiova
- Dobroęa
- Eforie
- Fagaras
- Giurgiu
- Hirsova
- Iasi
- Lugoj
- Mehadia
- Dobreta
- Craiova
- Sibiu
- Fagaras
- Timisoara
- Pitesti
- Sibiu
- Rimnicu Vilcea
- Vaslui
- Pitesti
- Zerind
- Giurgiu
- Neamt
- Oradea
- Ursiceni
- Bucharest

Straight-line distance to Bucharest:

- Arad: 366 km
- Bucharest: 0 km
- Craiova: 160 km
- Dobroęa: 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
- Ursiceni: 80 km
- Vaslui: 199 km
- Zerind: 374 km
A* tree search

Arad

366 = 0 + 366
A* tree search
A* tree search

- Arad
- Sibiu
  - Arad
  - Fagaras
  - Oradea
  - Rimnicu Vilcea
- Timisoara
- Zerind

Distances:
- Arad to Sibiu: 447 = 118 + 329
- Arad to Timisoara: 449 = 75 + 374
- Arad to Rimnicu Vilcea: 646 = 280 + 366
- Arad to Fagaras: 413 = 220 + 193
- Arad to Oradea: 415 = 239 + 176
- Arad to Zerind: 671 = 291 + 380

CMSC 421: Chapter 4, Sections 1–2  22
A* tree search
A* tree search
A* tree search
Optimality of $A^*$

Suppose some suboptimal goal $G_2$ has been generated and is in \textit{fringe}. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
\begin{align*}
  f(G_2) &= g(G_2) & \text{since } h(G_2) = 0 \\
  &> g(G_1) & \text{since } G_2 \text{ is suboptimal} \\
  &\geq f(n) & \text{since } h \text{ is admissible}
\end{align*}
\]

Since $f(G_2) > f(n)$, $A^*$ will never select $G_2$ for expansion.
**A* graph search**

A* can also be used with \texttt{Graph-Search}.

Optimality requirement is same as before:
\diamond The heuristic must be admissible

There are two completeness requirements:
\diamond One is the same as before: no infinite path has a finite cost
\diamond The other is something that Russell & Norvig don't mention:

Either the heuristic needs to be \textit{consistent},
or else we need to modify the \texttt{Graph-Search} algorithm
Consistent heuristics

Consistency is analogous to the *triangle inequality* from Euclidian geometry.

A heuristic is *consistent* if

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

If \( h \) is consistent, then for every child \( n' \) of \( n \),

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

I.e., \( f(n) \) is nondecreasing along any path.
Behavior of $A^*$ with consistent heuristic

If $h$ is consistent, then $A^*$ 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 $f_i < f_{i+1}$.
Behavior of A* with inconsistent heuristic

If $h$ is inconsistent, then

◊ As we go along a path, $f$ may sometimes decrease
◊ A* doesn’t always expand nodes in order of increasing $f$ value, because A* may find lower-cost paths to nodes it has already expanded
◊ A* will need to re-expand these nodes
◊ Problem: GRAPH-SEARCH won’t re-expand them
Behavior of A* with inconsistent heuristic

If $h$ is inconsistent, then

◊ As we go along a path, $f$ may sometimes decrease
◊ $A^*$ doesn’t always expand nodes in order of increasing $f$ value, because $A^*$ may find lower-cost paths to nodes it has already expanded
◊ $A^*$ will need to re-expand these nodes
◊ Problem: $\text{GRAPH-SEARCH}$ won’t re-expand them
Behavior of A* with inconsistent heuristic

If $h$ is inconsistent, then

◊ As we go along a path, $f$ may sometimes decrease
◊ A* doesn’t always expand nodes in order of increasing $f$ value, because A* may find lower-cost paths to nodes it has already expanded
◊ A* will need to re-expand these nodes
◊ Problem: GRAPH-SEARCH won’t re-expand them
Behavior of A* with inconsistent heuristic

If $h$ is inconsistent, then

◇ As we go along a path, $f$ may sometimes decrease
◇ A* doesn’t always expand nodes in order of increasing $f$ value, because A* may find lower-cost paths to nodes it has already expanded
◇ A* will need to re-expand these nodes
◇ Problem: GRAPH-SEARCH won’t re-expand them
Behavior of A* with inconsistent heuristic

If \( h \) is inconsistent, then

- As we go along a path, \( f \) may sometimes decrease
- A* doesn’t always expand nodes in order of increasing \( f \) value, because A* may find lower-cost paths to nodes it has already expanded
- A* will need to re-expand these nodes
- Problem: \textsc{Graph-Search} won’t re-expand them
Behavior of A* with inconsistent heuristic

If $h$ is inconsistent, then

◊ As we go along a path, $f$ may sometimes decrease
◊ A* doesn’t always expand nodes in order of increasing $f$ value, because A* may find lower-cost paths to nodes it has already expanded
◊ A* will need to re-expand these nodes
◊ Problem: GRAPH-SEARCH won’t re-expand them
A* for graphs

◊ Re-expands a node if it find a better path to the node
◊ Finds optimal solutions even if the heuristic is inconsistent

function $A^*$ (problem) returns a solution, or failure

  closed $\leftarrow$ an empty set
  fringe $\leftarrow$ a list containing $\text{MAKE-NODE}($Initial-State[problem$])$

loop do
  if fringe is empty then return failure
  node $\leftarrow$ REMOVE-FRONT(fringe)
  if $\text{GOAL-TEST}[problem]$ applied to $\text{STATE}(node)$ succeeds return node
  insert node into closed
  for each node $n \in \text{EXPAND}(node, problem)$ do
    if there is a node $m \in \text{closed} \cup \text{fringe}$ such that
    \[
    \text{STATE}(m) = \text{STATE}(n) \text{ and } f(m) \leq f(n)
    \]
    then do nothing
    else
      insert $n$ into fringe after the last node $m$ such that $f(m) \leq f(n)$
  end
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?** $O(\text{entire state space})$ in worst case, $O(d)$ in best case

**Space?**
Properties of A*

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

**Time?** $O(\text{entire state space})$ in worst case, $O(d)$ in best case

**Space?** Keeps all nodes in memory

**Finds optimal solutions?**
Properties of A*

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

Time? $O(\text{entire state space})$ in worst case, $O(d)$ in best case

Space? Keeps all nodes in memory

Finds optimal solutions? Yes

Additional properties:

A* expands all nodes in fringe that have $f(n) < C^*$

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

If $f$ is consistent, A* expands no nodes with $f(n) > C^*$
How to create admissible heuristics

◊ Suppose $P$ is a problem we’re trying to solve
Let $h^*(s) = \text{minimum cost of solution path}$

◊ Let $P'$ be a relaxation of $P$
Remove some constraints on what constitutes a solution

◊ Every solution path in $P$ is also a solution path in $P'$
$P'$ may have additional solution paths that aren’t solution paths in $P$

◊ Suppose we can find optimal solutions to $P'$ quickly
Let $h(s) = \text{minimum cost of solution path in } P'$
Then $h(s) \leq h^*(s)$, i.e., $h$ is an admissible heuristic for $P$
Example: Romania with step costs in km

\[ h(\text{at } c) = \text{cost of a straight line from city } c \text{ to Bucharest} \]

We relaxed the problem to allow paths that are straight lines.
Example: TSP

Well-known example: *traveling salesperson problem* (TSP)

- Given a *complete* graph (edges between all pairs of nodes)
- Find a least-cost *tour* (simple cycle that visits each city exactly once)

\[ \begin{align*}
\Rightarrow & \\
\Rightarrow & \\
\end{align*} \]

Relax the problem twice:

1. Let \{solutions\} include paths that visit all cities
2. Let \{solutions\} include trees

*Minimum spanning tree* can be computed in \(O(n^2)\)

\(\Rightarrow\) lower bound on the least-cost path that visits all cities
\(\Rightarrow\) lower bound on the least-cost tour
Example: the 8-puzzle

Relaxation 1: allow a tile to move to any other square regardless of whether the square is adjacent regardless of whether there’s another tile there already

This gives us $h_1(n) = \text{number of misplaced tiles}$

\[ h_1(S) = ? \]
Example: the 8-puzzle

Relaxation 1: allow a tile to move to any other square regardless of whether the square is adjacent regardless of whether there’s another tile there already.

This gives us $h_1(n) = \text{number of misplaced tiles}$

$h_1(S) = ? \quad 6$
Example: the 8-puzzle

Relaxation 2: allow a tile to move to any adjacent square, regardless of whether there’s another tile there already

This gives us $h_2(n) = \text{total Manhattan distance}$

Start State

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

Goal State

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

$h_2(S) = ?$
Example: the 8-puzzle

Relaxation 2: allow a tile to move to any adjacent square, regardless of whether there’s another tile there already

This gives us $h_2(n) = \text{total } \textit{Manhattan} \text{ distance}$

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

Start State \quad \text{Goal State}

$h_2(S) =? \quad 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$
Dominance

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

Notice that \( h_1(n) \leq h_2(n) \leq h^*(n) \) for all \( n \),
i.e., \( h_2 \) dominates \( h_1 \),

\( h_2 \)'s estimate of \( h^* \) is never worse than \( h_1 \)'s, and is often better than \( h_1 \)'s

Hence \( h_2 \) is better for search. Typical search costs:

\( d = 14 \)  \( \text{IDS} \approx 3,473,941 \text{ nodes} \)
\[ A^*(h_1) = 539 \text{ nodes} \]
\[ A^*(h_2) = 113 \text{ nodes} \]
\( d = 24 \)  \( \text{IDS} \approx 54,000,000,000 \text{ nodes} \)
\[ A^*(h_1) = 39,135 \text{ nodes} \]
\[ A^*(h_2) = 1,641 \text{ nodes} \]
One way to get dominance

If $h_a$ are $h_b$ admissible heuristic functions, then $h(n) = \max(h_a(n), h_b(n))$ is admissible and dominates $h_a, h_b$. 
Iterative-Deepening A*

function IDA*(problem) returns a solution

inputs: problem, a problem

\( f_0 \leftarrow h(\text{initial state}) \)

for \( i \leftarrow 0 \text{ to } \infty \) do

\( \text{result} \leftarrow \text{Cost-Limited-Search}(\text{problem}, f_i) \)

if result is a solution then return result

else \( f_{i+1} \leftarrow \text{result} \)

end

function Cost-Limited-Search(problem, fmax) returns solution or number

depth-first search, backtracking at every node \( n \) such that \( f(n) > f_{\text{max}} \)

if the search finds a solution then

\( \text{return} \) the solution

else

\( \text{return} \) min\( \{f(n) \mid \text{the search backtracked at } n\} \)
Properties of IDA*

Complete?
Properties of IDA* 

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

Time?
Properties of IDA* 

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

**Time?** Like A* if $f(n)$ is an integer and the number of nodes with $f(n) \leq k$ grows exponentially with $k$

**Space?**
Properties of IDA*  

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

**Time?** Like A* if \( f(n) \) is an integer and the number of nodes with \( f(n) \leq k \) grows exponentially with \( k \)  

**Space?** \( O(bd) \)  

**Optimal?**
Properties of IDA*  

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

**Time?** Like A* if $f(n)$ is an integer and the number of nodes with $f(n) \leq k$ grows exponentially with $k$

**Space?** $O(bd)$

**Optimal?** Yes

With consistent heuristic:

- IDA* cannot expand $f_{i+1}$ until $f_i$ is finished
- IDA* expands all nodes with $f(n) < C^*$
- IDA* expands no nodes with $f(n) \geq C^*$
Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy search expands lowest $h$
  – incomplete and not always optimal

A* search expands lowest $g + h$
  – complete, returns optimal solutions

IDA* is like a combination of A* and IDS
  – complete, returns optimal solutions
  – much lower space requirement than A*
  – same big-$O$ time if number of nodes grows exponentially with cost

Admissible heuristics can be derived from exact solution of relaxed problems