Informed Search and Exploration

Sections 3.5 and 3.6

Nilufer Onder
Department of Computer Science
Michigan Technological University
Outline

- Best-first search
- A* search
- Heuristics, pattern databases
- IDA* search
- (Recursive Best-First Search (RBFS), MA* and SMA* search)
Best-first search

- Idea: use an *evaluation function* for each node
- The evaluation function is an *estimate* of “desirability”
- Expand the most desirable unexpanded node
- The desirability function comes from domain knowledge
- Implementation:
  The *frontier* is a queue sorted in decreasing order of desirability
- Special cases:
  - greedy best first search
  - A* search
Romania with step costs in km

Sample straight line distances to Bucharest:
Greedy best-first search

- Evaluation function $h(n)$ (heuristic) = estimate of cost from $n$ to the closest goal
- E.g., $h_{SLD}(n) = \text{straight-line distance from } n \text{ to Bucharest}$
- Greedy best-first search expands the node that *appears* to be closest to goal
Greedy best-first search example
After expanding Arad

- Sibiu
  - 253
- Timisoara
  - 118
- Zerind
  - 75
- Arad
  - 140
After expanding Sibiu
After expanding Fagaras

The goal Bucharest is found with a cost of 450. However, there is a better solution through Pitesti \((h = 418)\).
Properties of greedy best-first search

- **Complete** No — can get stuck in loops
  For example, going from Iasi to Fagaras, Iasi → Neamt → Iasi → Neamt → ...
  Complete in finite space with repeated-state checking

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

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

- **Optimal** No
  (For example, the cost of the path found in the previous slide was 450. The path Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest has a cost of 140+80+97+101 = 418.)
A* search

- Idea: avoid expanding paths that are already expensive

- **Evaluation function** $f(n) = g(n) + h(n)$
  - $g(n)$ = *exact* 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

- A* search uses an *admissible* heuristic
  i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from $n$.
  (Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal $G$.)

- Straight line distance ($h_{SLD}(n)$) is an admissible heuristic because
  it never overestimates the actual road distance.
A* search example

366 = 0 + 366
After expanding Arad

Sibiu
393 = 140 + 253

Timisoara
447 = 118 + 329

Zerind
449 = 75 + 374
After expanding Sibiu

Arad

Fagaras

Oradea

Rimnicu V.

Sibiu

Timisoara

Zerind

646=280+366

415=239+176

671=291+380

447=118+329

449=75+374

413=220+193
After expanding Rimnicu Vilcea

- Arad
- Fagaras: 646 = 280 + 366
- Oradea: 415 = 239 + 176
- Sibiu: 671 = 291 + 380
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374
- Rimnicu V.: 526 = 366 + 160
- Pitesti: 417 = 317 + 100
- Sibiu: 553 = 300 + 253

Ch. 03b – p.15/51
Remember that the goal test is performed when a node is selected for expansion, not when it is generated.
After expanding Pitesti

- Arad
- Bucharest
- Oradea
- Sibiu
- Rimnicu V.
- Timisoara
- Zerind

**Arad**
- Fagaras

**Sibiu**
- Bucharest
- Oradea

**Rimnicu V.**
- Craiova
- Pitesti

**Bucharest**
- Craiova

**Craiova**
- Pitesti

**Pitesti**
- Sibiu

**Sibiu**
- Bucharest

Distance calculations:
- 646 = 280 + 366
- 591 = 338 + 253
- 450 = 450 + 0
- 671 = 291 + 380
- 526 = 366 + 160
- 553 = 300 + 253
- 418 = 418 + 0
- 615 = 455 + 160
- 607 = 414 + 193
- 447 = 118 + 329
- 449 = 75 + 374
Optimality of A* for trees

**Theorem:** A* search is optimal.

Note that, A* search uses an admissible heuristic by definition.

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 $G_1$. 
Optimality of A* for trees (cont’d)

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

\[ f(G_1) = g(G_1) \]
because \( h \) is 0 at a goal

\[ f(G_2) = g(G_2) \]
because \( h \) is 0 at a goal

\[ f(n) \leq f(G_1) \]
because \( h \) is admissible (never overestimates)

\[ f(G_1) < f(G_2) \]
because \( G_2 \) is suboptimal

\[ f(n) < f(G_2) \]
combine the above two

Since \( f(n) < f(G_2) \), A* will never select \( G_2 \) for expansion.
Progress of A* with an inconsistent heuristic

Note that $h$ is admissible, it never overestimates.
Progress of A* with an inconsistent heuristic

The root node was expanded. Note that f decreased from 6 to 4.
Progress of A* with an inconsistent heuristic

The suboptimal path is being pursued.
Progress of A* with an inconsistent heuristic

Goal found, but we cannot stop until it is selected for expansion.
Progress of $A^*$ with an inconsistent heuristic

The node with $f = 7$ is selected for expansion.
Progress of A* with an inconsistent heuristic

The optimal path to the goal is found.
A heuristic is *consistent* if
\[ h(n) \leq c(n, a, n') + h(n') \]

If \( h \) is consistent, we have

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

I.e., \( f(n) \) is nondecreasing along any path.
Optimality of A* for graphs

- **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 $f_i < f_{i+1}$

- With uniform-cost search (A* search with $h(n)=0$) the bands are “circular”.
  With a more accurate heuristic, the bands will stretch toward the goal and become more narrowly focused around the optimal path.
F-contours
Performance of A*

- The **absolute error** of a heuristic is defined as
  \[ \Delta \equiv h^* - h \]

- The **relative error** of a heuristic is defined as
  \[ \epsilon \equiv \frac{h^* - h}{h^*} \]

- Complexity with constant step costs: \( O(b^{\epsilon d}) \)

- Problem: there can be exponentially many states with \( f(n) < C^* \)
even if the absolute error is bounded by a constant
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 solution)
- **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 \( f(n) > C^* \)
E.g., for the 8-puzzle:

- $h_1(n) =$ number of misplaced tiles
- $h_2(n) =$ total *Manhattan* distance
  (i.e., no. of squares from desired location of each tile)

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Start State

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
```

Goal State

$h_1(S) = ??$

$h_2(S) = ??$
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:

- $d = 14$  
  IDS = 3,473,941 nodes
  $A^*(h_1) = 539$ nodes 
  $A^*(h_2) = 113$ nodes

- $d = 24$  
  IDS $\approx$ 54,000,000,000 nodes
  $A^*(h_1) = 39,135$ nodes 
  $A^*(h_2) = 1,641$ nodes
Effect of Heuristic on Performance

The effect is characterized by the effective branching factor \( b^* \)

- If the total number of nodes generated by \( A^* \) is \( N \) and
- the solution depth is \( d \),
- then \( b \) is branching factor of a uniform tree, such that

\[
N + 1 = 1 + b + (b)^2 + + (b)^d
\]

A well designed heuristic has a \( b \) close to 1.
Using relaxed problems to find heuristics

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.
- 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.
Relaxed problems (cont’d)

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

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour
Admissible heuristics can also be generated from the solution cost of sub-problems.

For example, in the 8-puzzle problem a sub-problem of getting the tiles 2, 4, 6, and 8 into position is a lower bound on solving the complete problem.

Pattern databases store the solution costs for all the sub-problem instances.

The choice of sub-problem is flexible: for the 8-puzzle a subproblem for 2, 4, 6, 8 or 1, 2, 3, 4 or 5, 6, 7, 8, . . . could be created.
Iterative Deepening A* (IDA*)

- Idea: perform iterations of DFS. The cutoff is defined based on the $f$-cost rather than the depth of a node.
- Each iteration expands all nodes inside the contour for the current $f$-cost, peeping over the contour to find out where the contour lies.
function IDA* (*problem*)
returns a solution sequence

inputs: *problem*, a problem
local variables:
   *f-limit*, the current *f*-COST limit
   *root*, a node

*root* ← MAKE-NODE(INITIAL-STATE[*problem*])
*f-limit* ← *f*-COST(*root*)
loop do
   *solution*, *f-limit* ← DFS-CONTOUR(*root*, *f-limit*)
   if *solution* is non-null then return *solution*
   if *f-limit* = ∞ then return failure
Iterative Deepening A* (IDA*)

function DFS-CONTOUR (node, f-limit)
  returns a solution sequence and a new \( f \)-COST limit

inputs: \( node \), a node
        \( f\text{-}{\text{limit}} \), the current \( f\text{-}{\text{COST}} \) limit

local variables:
  \( next-f \), the \( f\text{-}{\text{COST}} \) limit for the next contour, initially \( \infty \)

if \( f\text{-}{\text{COST}}[node] > f\text{-}{\text{limit}} \) then return null, \( f\text{-}{\text{COST}}[node] \)
if Goal-Test[problem](STATE[node]) then return node, f-limit
for each node \( s \) in SUCCESSORS(node) do
  solution, new-f ← DFS-CONTOUR(s, f-limit)
  if solution is non-null then return solution, f-limit
  next-f ← MIN(next-f, new-f)
return null, next-f
How would IDA* proceed?

The blue nodes are the ones A* expanded. For IDA*, they define the new f-limit.
Properties of IDA*

- **Complete** Yes, similar to A*.
- **Time** Depends strongly on the number of different values that the heuristic value can take on. 8-puzzle: few values, good performance TSP: the heuristic value is different for every state. Each contour only includes one more state than the previous contour. If A* expands $N$ nodes, IDA* expands $1 + 2 + \ldots + N = O(N^2)$ nodes.
- **Space** It is DFS, it only requires space proportional to the longest path it explores. If $\delta$ is the smallest operator cost, and $f^*$ is the optimal solution cost, then IDA* will require $bf^*/\delta$ nodes.
- **Optimal** Yes, similar to A*
Recursive Best-First Search (RBFS)

- Idea: mimic the operation of standard best-first search, but use only linear space
- Runs similar to recursive depth-first search, but rather than continuing indefinitely down the current path, it uses the *f-limit* variable to keep track of the best alternative path available from any ancestor of the current node.
- If the current node exceeds this limit, the recursion unwinds back to the alternative path. As the recursion unwinds, RBFS replaces the *f-value* of each node along the path with the best *f-value* of its children. In this way, it can decide whether it’s worth reexpanding a forgotten subtree.
function Recursive-Best-First-Search (problem)
returns a solution or failure
return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), \infty)
function RBFS (problem, node, f-limit)
returns a solution or failure and a new f-cost limit
if problem.Goal-Test(node.State) then return Solution(node)
successors ← []
for each action in problem.Actions(node.State) do
    add Child-Node(problem, node, action) into successors
if successors is empty then return failure, ∞
for each s in successors do
    /* update f with value from previous search, if any */
    s.f ← max (s.g + s.h, node.f)
loop do
    best ← the lowest f-value in successors
    if best.f > f-limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS (problem, best, min(f-limit,alternative))
    if result ≠ failure then return result

Progress of RBFS

(a) After expanding Arad, Sibiu, Rimnicu Vilcea
(b) After unwinding back to Sibiu and expanding Fagaras
(c) After switching back to Rimnicu Vilcea and expanding Pitesti
Stage (a): The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras).

Stage (b): The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best value of 450.

Stage (c): The recursion unwinds and the best value of the of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path through Timisoara costs at least 447, the expansion continues to Bucharest.
Properties of RBFS

- **Complete** Yes, similar to A*.

- **Time** The time complexity is difficult to characterize: it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded. Each mind change corresponds to an iteration of IDA*, and could require many reexpansions of forgotten nodes to recreate the best path and extend it one more node. RBFS is somewhat more efficient than IDA*, but still suffers from excessive node regeneration.
Space  IDA* and RBFS suffer from using too little memory. Between iterations, IDA* retains only a single number: the current \( f \)-cost limit. RBFS retains more information in memory, but only uses \( O(bd) \) memory. Even if more memory is available, RBFS has no way to make use of it.

Optimal  Yes, similar to A*.
MA* and SMA*

Idea: use all the available memory
IDA* remembers only the current $f$-cost limit
RBFS uses linear space

Proceeds just like A*, expanding the best leaf until the memory is full. When the memory if full, drops the worst leaf node.
The evaluation function for a node $n$ is:

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

- If only $g(n)$ is used, we get uniform-cost search
- If only $h(n)$ is used, we get greedy best-first search
- If both $g(n)$ and $h(n)$ are used we get best-first search
- If both $g(n)$ and $h(n)$ are used with an admissible heuristic we get $A^*$ search
- A consistent heuristic is admissible but not necessarily vice versa
Summary (cont’d)

- Admissibility is sufficient to guarantee solution optimality for tree search.
- Consistency is required to guarantee solution optimality for graph search.
- If an admissible but not consistent heuristic is used for graph search, we need to adjust path costs when a node is rediscovered.
- Heuristic search usually brings dramatic improvement over uninformed search.
- Keep in mind that the f-contours might still contain an exponential number of nodes.