A* Search

- CSCE 420 guest lecture
- Yoonsuck Choe
- September 13, 2016

Recap: General Search Algorithm

Pseudo-code:

```
function General-Search (problem, Que-Fn)
  node-list := initial-state
  loop begin
      // fail if node-list is empty
      if Empty(node-list) then return FAIL
      // pick a node from node-list
      node := Get-First-Node(node-list)
      // if picked node is a goal node, success!
      if (node == goal) then return as SOLUTION
      // otherwise, expand node and enqueue
      node-list := Que-Fn(node-list, Expand(node))
  loop end
```


Recap: Evaluation of Search Strategies

- time-complexity: how many nodes visited so far?
- space-complexity: how many nodes must be stored in node-list at any given time?
- completeness: if solution exists, guaranteed to be found?
- optimality: guaranteed to find the best solution?

Recap: Best First Search

function Best-First-Search (*problem, Eval-Fn*)

Queuing-Fn $\leftarrow$ sorted list by *Eval-Fn*(node)
return General-Search(*problem, Queuing-Fn*)

- The queuing function queues the expanded nodes, and sorts it every time by the *Eval-Fn* value of each node.
- One of the simplest Eval-Fn: **estimated cost** to reach the goal.

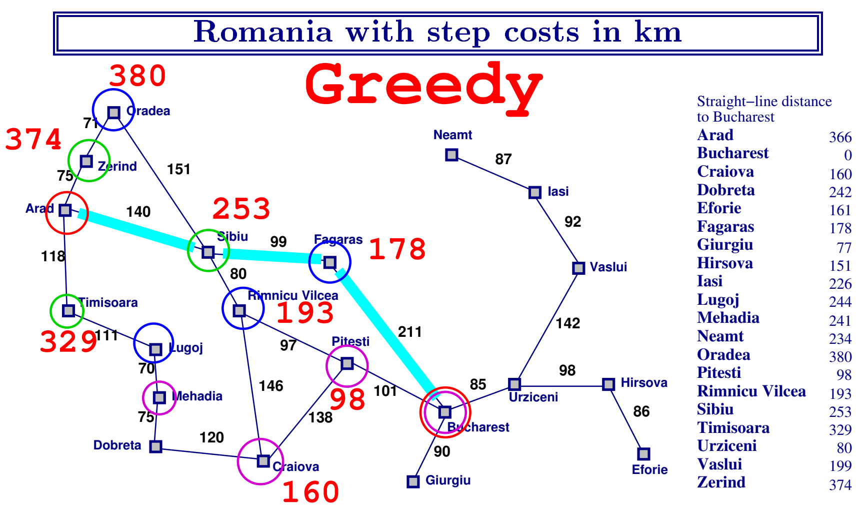
Recap: Heuristic Function

- $h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state.
- The only requirement is the $h(n) = 0$ at the goal.
- **Heuristics** means “to find” or “to discover”, or more technically, “how to solve problems” (Polya, 1957).

Recap: Greedy Best-First Search

> **function** Greedy-Best-First Search (*problem*)
>
> $h(n)$=estimated cost from n to goal
>
> **return** Best-First-Search(*problem*,h)

- Best-first with heuristic function $h(n)$



Total Path Cost = 450

A^*: Uniform Cost + Heuristic Search

Avoid expanding paths that are already found to be expensive:

- $f(n) = g(n) + h(n)$
- $f(n)$: estimated cost to goal through node n
- **provably complete and optimal!**
- **restrictions**: $h(n)$ should be an **admissible heuristic**
- admissible heuristic: one that **never overestimate** the actual cost of the best solution through n

A^* Search

function A^*-Search (*problem*)

$g(n)$=current cost up till n

$h(n)$=estimated cost from n to goal

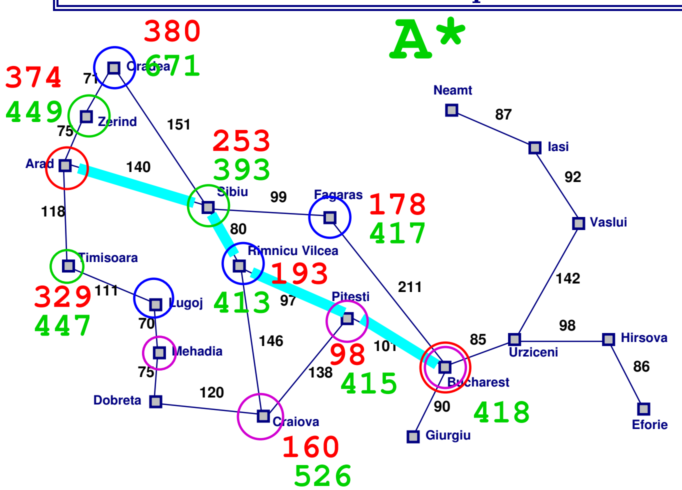
return Best-First-Search(*problem*,$g + h$)

- Condition: $h(n)$ must be an **admissible heuristic function**!
- A^* is **optimal!**

Behavior of A^* Search

- usually, the f value never decreases along a given path: **monotonicity**
- in case it is nonmonotonic, i.e. $f(Child) < f(Parent)$, make this adjustment: $f(Child) = max(f(Parent), g(Child) + h(Child))$.
- this is called **pathmax**

Romania with step costs in km



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Total Path Cost = 418

Optimality of A^*

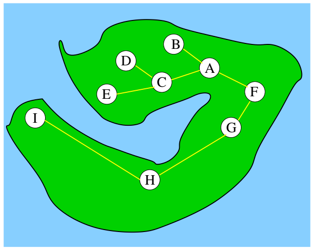
G_2: suboptimal goal in the node-list.

n: unvisited node on a shortest path to goal G_1

- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $> g(G_1)$ since G_2 is suboptimal
- $\geq f(n)$ since h is admissible

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

Optimality of A^*: Example



1. **Expansion of parent disallowed**: search fails at nodes **B**, **D**, and **E**.
2. **Expansion of parent allowed**: paths through nodes **B**, **D**, and **E** with have an inflated path cost $g(n)$, thus will become nonoptimal.

$$\underbrace{A \to C \to E \to C \to}_{\text{inflated path cost}} A \to F \to \ldots$$

Lemma to Optimality of A^*

Lemma: A^* visits nodes in order of increasing $f(n)$ value.

- Gradually adds **f-contours** of nodes (cf. BFS adds layers).
- The goal state may have a f value: let's call it f^*
- This means that all nodes with $f < f^*$ will be visited!

Complexity of A^*

A^* is complete and optimal, but space complexity can become exponential if the heuristic is not good enough.

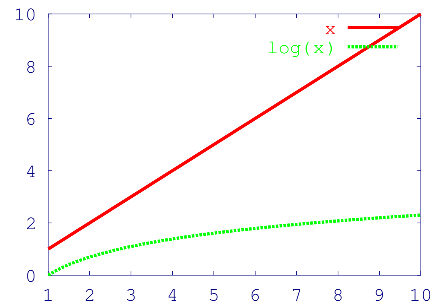
- condition for **subexponential** growth:

  $$|h(n) - h^*(n)| \leq O(\log h^*(n)),$$

  where $h^*(n)$ is the **true** cost from n to the goal.
- that is, error in the estimated cost to reach the goal should be less than even linear, i.e. $< O(h^*(n))$.

Unfortunately, with most heuristics, error is at least proportional with the true cost, i.e. $\geq O(h^*(n)) > O(\log h^*(n))$.

Linear vs. Logarithmic Growth Error



- Error in heuristic: $|h(n) - h^*(n)|$.
- For most heuristics, the error is at least linear.
- For A^* to have subexponential growth, the error in the heuristic should be on the order of $O(\log h^*(n))$.

Problem with A^*

Space complexity is usually **exponential**!

- we need a memory bounded version
- one solution is: Iterative Deepening A^*, or IDA^*

A^*: Evaluation

- Complete : unless there are infinitely many nodes with $f(n) \leq f(G)$
- Time complexity: exponential in (relative error in h $\times$ length of solution)
- Space complexity: same as time (keep all nodes in memory)
- Optimal

Heuristic Functions: Example

Eight puzzle

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

  $h_1(n)$ = 7 (not counting the blank tile)
  $h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.

Dominance

If $h_2(n) \geq h_1(n)$ for all n and both are admissible, then we say that $h_2(n)$ **dominates** $h_1(n)$, and is better for search.

Typical search costs for depth $d = 14$:

- Iterative Deepening : 3,473,941 nodes visited
- $\mathrm{A}^*(h_1)$: 539 nodes
- $\mathrm{A}^*(h_2)$: 113 nodes

Observe that in A^*, every node with $f < f^*$ is visited. Since $f = g + h$, nodes with $h(n) < f^* - g(n)$ will be visited, so larger h will result in less nodes being visited.

- f^* is the f value for the optimal solution path.

Designing Admissible Heuristics

Relax the problem to obtain an admissible heuristics.

For example, in 8-puzzle:

- allow tiles to move anywhere $\rightarrow h_1(n)$
- allow tiles to move to any adjacent location $\rightarrow h_2(n)$

For traveling:

- allow traveler to travel by air, not just by road: **SLD**

Other Heuristic Design

- Use composite heuristics: $h(n) = max(h_1(n), ..., h_m(n))$
- Use statistical information: random sample h and true cost to reach goal. Find out how often h and true cost is related.

Optional: Iterative Deepening A^*: IDA^*

A^* is complete and optimal, but the performance is limited by the available space.

- Basic idea: only search within a certain f bound, and gradually increase the f bound until a solution is found.
- More on IDA^* next time.

IDA^*

function IDA^*(*problem*)

root $\leftarrow$ Make-Node(Initial-State(*problem*))

f-limit $\leftarrow$ f-Cost(*root*)

loop do

solution, f-limit $\leftarrow$ DFS-Contour(*root, f-limit*)

if *solution* != NULL **then return** *solution*

if *f-limit* == ∞ **then return** *failure*

end loop

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

DFS-Contour(*root, f-limit*)

Find solution from node **root**, within the f-cost limit of **f-limit**.
DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return fail.
- if **root** is a goal node, return solution and new f-cost limit.
- **recursive call** on all successors and return solution and **minimum f-limit** returned by the calls
- return **null solution** and new **f-limit** by default

Similar to the recursive implementation of DFS.

IDA^*: Evaluation

- complete and optimal (with same restrictions as in A^*)
- space: proportional to longest path that it explores (because it is depth first!)
- time: dependent on the number of different values $h(n)$ can assume.

IDA^*: Time Complexity

Depends on the heuristics:

- small number of possible heuristic function values $\longrightarrow$ small number of f-contours to explore $\longrightarrow$ becomes similar to A^*
- complex problems: each f-contour only contain one new node

  if A^* expands N nodes,
  IDA^* expands
  $1 + 2 + .. + N = \frac{N(N+1)}{2} = O(N^2)$
- a possible solution is to have a **fixed** increment ϵ for the f-limit $\longrightarrow$ solution will be suboptimal for at most ϵ (ϵ-admissible)