Artificial Intelligence

Heuristic Search

Christopher Simpkins

KENNESAW STATE UNIVERSITY

Informed (Heuristic) Search Strategies

- Use domain-specific hints about "distance" from goals
- Hints encapsulated in a **heuristic function**, $h(node)$:
  - $h(node)$ = estimated cost of cheapest path from $node$ to a goal state
  - h is really a function of $state$, not $node$. We use $h(node)$ to be consistent with $f(node)$ in best-first search, and path cost, $g(node)$.
  - Book uses $f(n)$, $g(n)$ and $h(n)$. I use $node$ instead of n to clearly distinguish from n as an index in problem size, N.

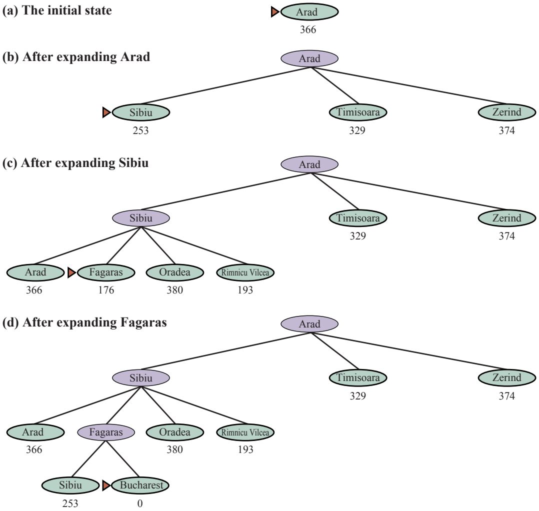
Example Heuristic for Romania, h_{SLD}:

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

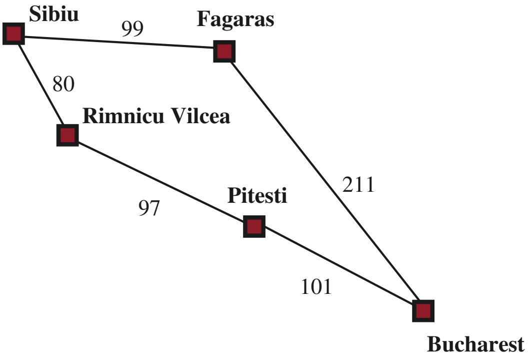
Straight line distances to Bucharest from each of the cities in Romania.

Greedy Best-First Search

- Recall that best-first search uses a priority queue for its frontier, ordered by $f(node)$
- Greedy best-first search uses $f(node) = h(node)$
- Greediness: get as close to the goal as possible in each step



Optimality of Greedy Best-First Search



- Greedy best-first search returns the path via Sibiu and Fagaras to Bucharest.
- The path through Rimnicu Vilcea and Pitesti is 32 miles shorter.

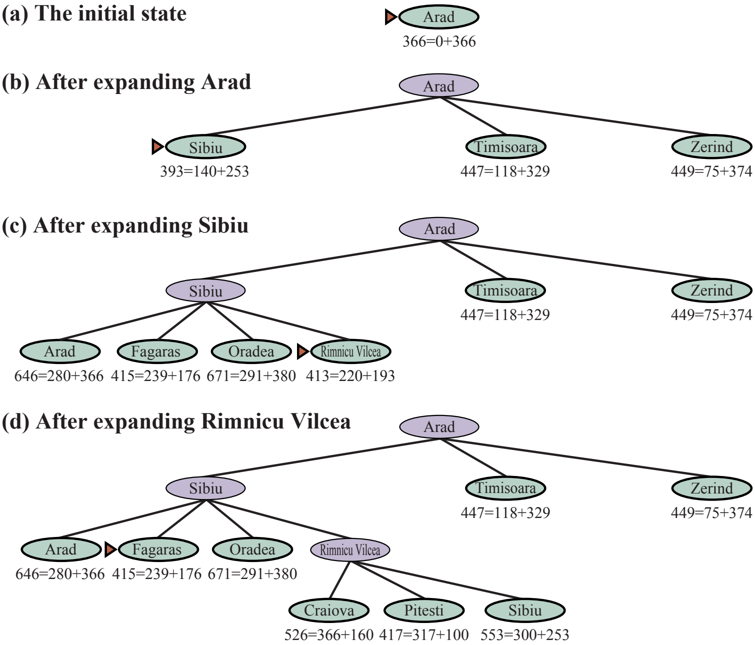
A^* Search

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

- Complete
- Optimal with an admissible heuristic
- Relatively efficient, but can generate exponential number of nodes for some problems.

Heavily dependent on quality of heuristic function.

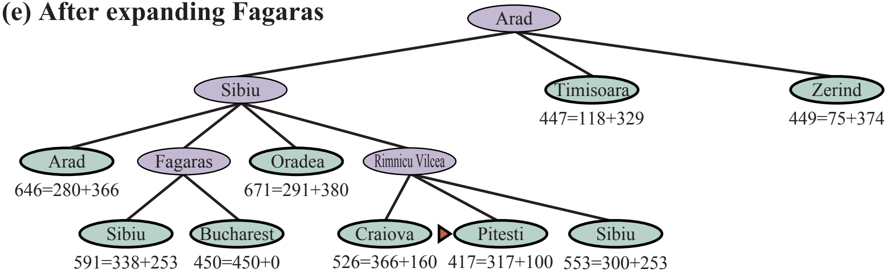
A^* Progress Part 1



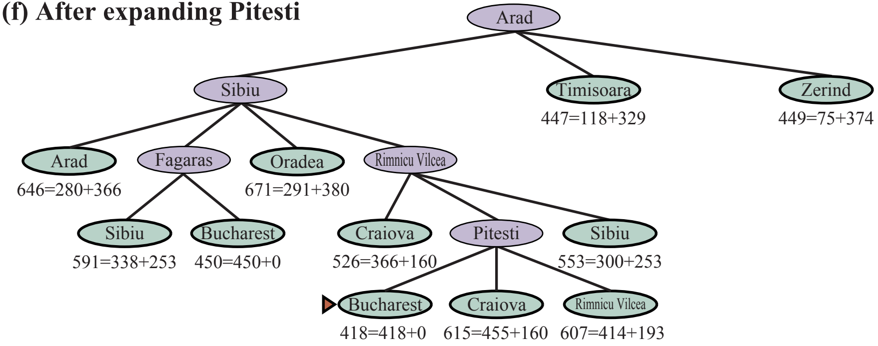
KENNESAW STATE UNIVERSITY

A^* Progress Part 2

(e) After expanding Fagaras

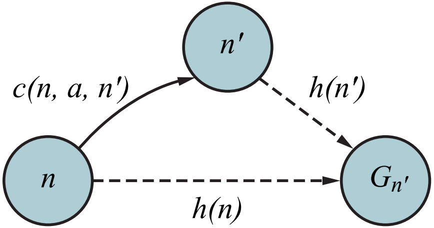


(f) After expanding Pitesti



Admissibility and Consistency

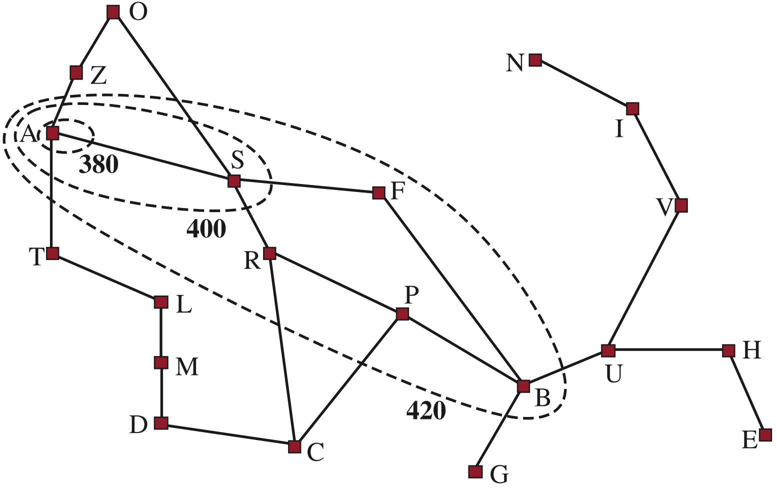
- An **admissible** heuristic never overestimates the cost to reach a goal.
- A **consistent** heuristic is a kind of local admissibility: for every node $node$ and successor $node'$ generated by action a: $h(node) \leq c(node, a, node') + h(n')$. This is a form of **triangle inequality**.



- Admissibility is required to guarantee cost-optimality in A^*.
- Consistency improves performance by guaranteeing that the first time we reach a node, it is on the optimal path – so we don't re-evaluate multiple paths to the same node.

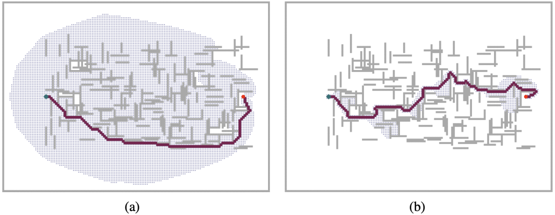
Search Contours

- In a topographical map, countours indicate a constant elevation
- In a search contour of a state space, a contour indicates an upper bound on path cost in a region
  - In the 400 countour, each node has $f(node) = g(node) + h(node) \leq 400$.



Satisficing Search: A^* vs Weighted A^*

- Detour index: multiplier applied to straight-line distance to account for curvature of roads. E.g., detour index of 1.3 means a road connecting locations 10 miles apart would be estimated as 13 miles long.
- Weighted A^* search: apply a wieght, like detour index, to $h(node)$
  - $f(n) = g(n) + w \cdot h(n)$, for some $w > 1$
- Results in inadmissible heuristic (overestimates), but can improve search speed.



(a) an A^* search and (b) a weighted A^* search with weight $w = 2$.

- The gray bars are obstacles, the purple line is the path from the green start to red goal, and the small dots are states that were reached by each search.
- On this particular problem, weighted A^* explores 7 times fewer states and finds a path that is 5% more costly.

Memory-Bounded Search

A^* is not memory-efficient. Some approaches to imporoving memory efficiency:

- **Beam search** keeps only the k nodes with lowest f values.
  - Forms a narrow "beam" through the search space.
  - Not complete or optimal, but good enough with sufficiently large k
  - Alternative: keep nodes within σ of best f score, so only narrow beam when there are clearly better nodes.
- **Iterative-deepening** A^* uses $f = g + h$ as the cut-off for the frontier instead of depth.
  - Iteratively expands contours of search space.
- **Recursive best-first search** resembles depth-first search.
  - Instead of continuing down a path indefinitely, keeps track of path with second best f value of ancestor. If that f value is exceeded, discards path and backs up to the alternative path.
  - f value of discarded path is kept in case alternative doesn't work out.
- **Simplified memory-bounded** A^* **(**SMA^***)** is similar to RBFS, but expands best leaf node until memory is full. Then it discards the worst leaf and continues.

KENNESAW STATE UNIVERSITY

Heuristic Functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- Misplaced tiles, $h_1 = 8$.
- Manhattan distance, $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

True solution cost is 26, so neither heristic overestimates.

Heuristic Accuracy and Performance

- Effective branching factor, b^*: for N nodes, branching factor of uniform tree of depth d that would contain $N+1$ nodes. Want close to 1.
  - $N+1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

- h_2 dominates h_1 because for any $node$, $h_2(node) \geq h_1(node)$
- We want a heuristic that underestimates, but by as little as possible

Designing Heuristic Functions

- Relaxing the problem definition
- Storing precomputed solution costs for subproblems in a pattern database
- Defining landmarks
- Learning from experience

Designing heuristic functions requires domain knowledge. But there is an automated approach based on relaxed problem definitions: `Absolver`