Foundations of Artificial Intelligence

Informed Search

CS472 – Fall 2007
Thorsten Joachims

Informed Methods: Heuristic Search

Idea: Informed search by using problem-specific knowledge.

Best-First Search: Nodes are selected for expansion based on an *evaluation function*, *f*(*n*). Traditionally, *f* is a cost measure.

Heuristic: Problem specific knowledge that (tries to) lead the search algorithm faster towards a goal state. Often implemented via *heuristic function h(n).*

→ Heuristic search is an attempt to search the most promising paths first. Uses heuristics, or rules of thumb, to find the best node to expand next.

Generic Best-First Search

1. Set *L* to be the initial node(s) representing the initial state(s).
2. If *L* is empty, fail. Let *n* be the node on *L* that is ``most promising'' according to *f*. Remove *n* from *L*.
3. If *n* is a goal node, stop and return it (and the path from the initial node to *n*).
4. Otherwise, add *successors*(*n*) to *L*. Return to step 2.

Greedy Best-First Search

Heuristic function *h(n):* estimated cost from node n to nearest goal node.

Greedy Search: Let *f(n) = h(n).*

Example: 8-puzzle

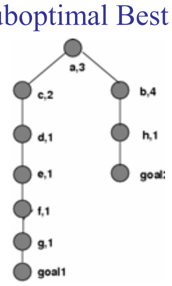
5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

Example: Suboptimal Best First-Search



There exist strategies that enable optimal paths to be found without examining all possible paths.

A* Search

Idea: Use total estimated solution cost:

g(*n*): Cost of reaching node n̲ from initial node

h(*n*): Estimated cost from node n to nearest goal

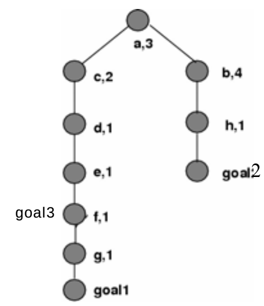
A* evaluation function: *f*(*n*) = *g*(*n*) + *h*(*n*)
→ *f*(*n*) is estimated cost of cheapest solution through n.

Comparison of Search Costs on 8-Puzzle

h1: number of misplaced tiles
h2: Manhattan distance

	Search Cost		
d	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
16	-	1301	211
18	-	3056	363
20	-	7276	676
22	-	18094	1219
24	-	39135	1641

Example



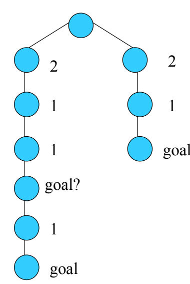
Admissibility

$h^*(n)$ *Actual* cost to reach a goal from n.

Definition: A heuristic function h is **optimistic** or **admissible** if $h(n) \leq h^*(n)$ for all nodes n. (h **never overestimates** the cost of reaching the goal.)

Theorem: If h is admissible, then the A* algorithm will never return a suboptimal goal node.

Example



8-Puzzle

1. h_C = number of misplaced tiles
2. h_M = Manhattan distance

Which one should we use?

$$h_C \leq h_M \leq h^*$$

Comparison of Search Costs on 8-Puzzle

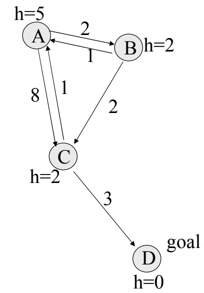
h1: number of misplaced tiles
h2: Manhattan distance

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Constructing Admissible Heuristics

- Use an admissible heuristic derived from a **relaxed version** of the problem.
- Use information from **pattern databases** that store exact solutions to subproblems of the problem.
- Use **inductive learning** methods.

Example: A*



Proof of the optimality of A*

Assume: h admissible; f non-decreasing along any path.

Proof [Optimality of A*]:
Let G be an optimal goal state, with path cost f^*.
Let G_2 be a suboptimal goal state, with path cost $g(G_2) > f^*$.
Let n is a node on an optimal path to G.

Assume that G_2 is expanded before n:
• Because h is admissible, we must have

$$f^* \geq f(n).$$

• If n is not expanded before expanding G_2, we must have

$$f(n) \geq f(G_2).$$

• Gives us $f^* \geq f(G_2) = g(G_2)$.
• Contradiction to G_2 suboptimal!

Proving the optimality of *A**

Lemma: If h is admissible, then f=g+h can be made non decreasing.

1. g is non-decreasing since cost positive.
2. But h can be increasing, while still admissible.
   Example: Node p, with $f=3+4=7$; child n, with $f=4+2=6$.
3. But because any path through n is also a path through p, we can see that the value *6* is meaningless, because we already know the true cost is at least *7* (because h is admissible).
4. So, make $f = max(f(p), g(n)+h(n))$

A*

Optimal: yes

Complete: Unless there are infinitely many nodes with f(n)<f*.
Assume locally finite:
(1) finite branching, (2) every operator costs at least $\delta > 0$

Complexity (time and space): Still exponential because of breadth-first nature. Unless $|h(n) - h^*(n)| \leq O(log(h^*(n)))$, with h^* true cost of getting to goal.

A* is **optimally efficient**: given the information in h, no other optimal search method can expand fewer nodes.

IDA*

Memory is a problem for the A* algorithms.

IDA* is like iterative deepening, but uses an f-cost limit rather than a depth limit.

At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

Each iteration uses conventional depth-first search.

Recursive best-first search (RBFS)

Similar to a DFS, but keeps track of the *f*-value of the best alternative path available from any ancestor of the current node.

If current node exceeds this limit, recursion unwinds back to the alternative path, replacing the *f*-value of each node along the path with the best *f*-value of its children.

(RBFS remembers the *f*-value of the best leaf in the forgotten subtree.)

SMA*

Simplified Memory-Bounded A* Search:

- While memory available, proceeds just like A*, expanding the best leaf.
- If memory is full, drops the **worst** leaf node - the one the highest *f*-cost; and stores this value in its parent node.

(Won't know which way to go from this node, but we will have some idea of how worthwhile it is to explore the node.)