Informed Search

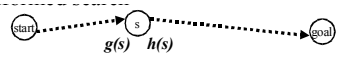
Louis Oliphant
cs540 section 2
Computer Sciences Department
University of Wisconsin, Madison

[Based on slides from Andrew Moore and Jerry Zhu http://www.cs.cmu.edu/~awm/tutorials]

Uninformed vs. informed search

- Uninformed search (BFS, uniform-cost, DFS, IDS etc.)
  - Knows the actual cost ***g(s)*** from start to an expanded node ***s***, but that's it.

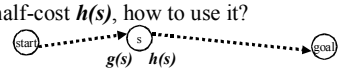
Heuristic – Of or relating to a usually speculative formulation serving as a guide in the investigation or solution of a problem. (dictionary.com)



  - also has a heuristic ***h(s)*** of the cost from ***s*** to goal. ('h'= heuristic, non-negative)
  - Can be much faster than uninformed search.

Recall: Uniform-cost search

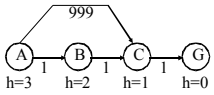
- Uniform-cost search: uninformed search when edge costs are not the same.
- Complete (will find a goal). Optimal (will find the least-cost goal).
- Always expand the node with the least ***g(s)***
  - Use a priority queue:
    - Push in states with their first-half-cost ***g(s)***
    - Pop out the state with the least ***g(s)*** first.
- Now we have an estimate of the second-half-cost ***h(s)***, how to use it?



Best-first greedy search

- Idea 1: use ***h(s)*** instead of ***g(s)***
- Always expand the node with the least ***h(s)***
  - Use a priority queue:
    - Push in states with their first-half-cost ***h(s)***
    - Pop out the state with the least ***h(s)*** first.
- Known as "best first greedy" search
- How's this idea?

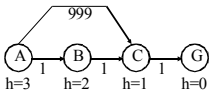
Best-first greedy search looking stupid



- It will follow the path $A \rightarrow C \rightarrow G$ (why?)
- Obviously not optimal

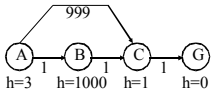
A search

- Idea 2: use ***g(s)+h(s)***
- Always expand the node with the least ***g(s)+h(s)***
  - Use a priority queue:
    - Push in states with their first-half-cost ***g(s)+h(s)***
    - Pop out the state with the least ***g(s)+h(s)*** first.
- Known as "A" search
- How's this idea?



- Works for this example

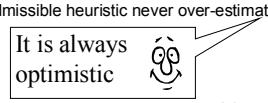
A search



- Still not quite right… A search is not optimal.

A* search

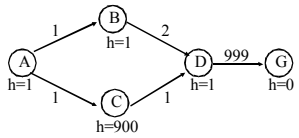
- Same as A search, but the heuristic function ***h()*** has to satisfy ***h(s)*** $\leq$ ***h*(s)***, where ***h*(s)*** is the true cost from node *s* to the goal, for all *s*.
- Such heuristic function ***h()*** is called admissible.
  - An admissible heuristic never over-estimates



- A search with admissible ***h()*** is called A* search.

A* revisiting expanded states

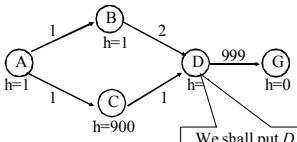
- One complication: A* can revisit an expanded state, and discover a shorter path



- Can you find the state in question?

A* revisiting expanded states

- One complication: A* can revisit an expanded state, and discover a shorter path

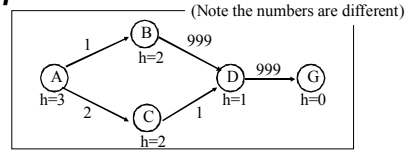


We shall put *D* back into the priority queue, with the smaller *g+h*

- Can you find the state in question?

What if A* revisits a state in the queue?

(Note the numbers are different)



- We've seen this before, with uniform cost search
- 'promote' *D* in the queue with the smaller cost

The A* algorithm

1. Put start *s* into priority queue *Q* with *f(s)=g(s)+h(s)*
2. Q empty? Sadly stop and admit failure
3. Remove node *n* with the lowest *f(n)* from *Q*
4. *n* is a goal? Happily stop and report success
5. Expand *n*: FOR each *n'* in **successors**(*n*):
   1. Let *f(n')=g(n')+h(n')=g(n)+cost(n,n')+h(n')*
   2. IF *n'* not seen before OR *n'* previously expanded with a higher *f* OR *n'* currently in *Q* with a higher *f*
   3. THEN place/promote *n'* on priority *Q* with *f(n')*
   4. ELSE ignore *n'*
6. Goto 2.

g(s)=0 at start

A* is optimal

- Suppose A* finds a suboptimal path ending in goal G', where $f(G') > f^*$ = cost of optimal path
- Let's look at the first unexpanded node n on the optimal path (n exists, otherwise the optimal goal would have been found)
- $f(n) > f(G')$, otherwise we would have expanded n
- $f(n) = g(n)+h(n)$ by definition
  $= g^*(n)+h(n)$ because n is on the optimal path
  $\leq g^*(n)+h^*(n)$ because h is admissible
  $= f^*$ because n is on the optimal path
- $f^* \geq f(n) > f(G')$, contradicting the assumption at top

Admissible heuristic functions h

- 8-puzzle example

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

- Which of the following are admissible heuristics?
  - h(n)=number of tiles in wrong position
  - h(n)=0
  - h(n)=1
  - h(n)=sum of Manhattan distance between each tile and its goal location

Admissible heuristic functions *h*

- 8-puzzle example

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

- Which of the following are admissible heuristics?
  - h(n)=number of tiles in wrong position YES
  - h(n)=0 YES, uninformed uniform cost search
  - h(n)=1 NO, goal state
  - h(n)=sum of Manhattan distance between each tile and its goal location YES

Admissible heuristic functions h

- In general, which of the following are admissible heuristics? h*(n) is the true optimal cost from n to goal.
  - h(n)=h*(n)
  - h(n)=max(2,h*(n))
  - h(n)=min(2,h*(n))
  - h(n)=h*(n)-2
  - h(n)=sqrt(h*(n))

Admissible heuristic functions *h*

- In general, which of the following are admissible heuristics? h*(n) is the true optimal cost from n to goal.

  •h(n)=h*(n) YES

  •h(n)=max(2,h*(n)) NO

  •h(n)=min(2,h*(n)) YES

  •h(n)=h*(n)-2 NO, possibly negative

  •h(n)=sqrt(h*(n)) NO if h*(n)<1

Heuristics for Admissible heuristics

- How to construct heuristic functions?

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

- Often by relaxing the constraints
  - •h(n)=number of tiles in wrong position
    Allow tiles to fly to their destination in one step
  - •h(n)=sum of Manhattan distance between each tile and its goal location
    Allow tiles to move on top of other tiles

"my heuristic is better than yours"

- A heuristic function h2 **dominates** h1 if for all s

  $h1(s) \le h2(s) \le h^*(s)$
- We prefer heuristic functions as close to h* as possible, but not over h*.

But

- Good heuristic function might need complex computation
- Time may be better spent, if we use a faster, simpler heuristic function and expand more nodes

The Value of Good Heuristics

8-puzzle with :
ids – iterative deepening search
h1 – number of tiles in wrong position
h2 – Manhattan distance of tiles

	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

A*: the dark side

- A* can use lots of memory.
  O(number of states)
- For large problems A* will run out of memory
- We'll look an alternative:
  - IDA*

IDA*: iterative deepening A*

- Memory bounded search. Assume integer costs
  1. Do path checking DFS, do not expand any node with $f(n)>0$. Stop if we find a goal.
  2. Do path checking DFS, do not expand any node with $f(n)>1$. Stop if we find a goal.
  3. Do path checking DFS, do not expand any node with $f(n)>2$. Stop if we find a goal.
  4. Do path checking DFS, do not expand any node with $f(n)>3$. Stop if we find a goal.
  5. … repeat this, increase threshold by 1 each time until we find a goal.
- This is complete, optimal, but more costly than A* in general.

What you should know

- Know why best-first greedy search is bad.
- Thoroughly understand A*
- Trace simple examples of A* execution.
- Understand admissible heuristics.

A* is complete

- If branching factor is finite, and edge cost bounded away from zero
- There are finitely many acyclic paths in the search tree
- A* only ever considers acyclic paths
- On each iteration of A* a new acyclic path is generated because:
  - When a node is added the first time, a new path exists.
  - When a node is "promoted", a new path to that node exists. It must be new because it's shorter.
- So the very most work it could do is to look at every acyclic path in the graph.
- So, it terminates.