START LECT WED JAN 23

Heuristic Search

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter, 2019

Outline

Motivation, Definition
A* Algorithm
Admissibility and Consistency
Heuristics for the 8 Puzzle
Designing Heuristics

Motivation

DFS, BFS, IDDFS; do not consider qualities of states on their search
only keeps checking if they are at a goal state

- Blind search can waste time and space. (due to the combinatorial explosion.)
- Additional knowledge MAY be available - - how could it help?
  - E.g., finding a shortest route from Wash. U. to U. Wash. (St. Louis to Seattle).
  - Blind search considers all directions equally, including towards Wash. D.C.
  - Additional knowledge: we need to head northwest.

using additional knowledge to make INFORMED decisions about which
states to transition to next; BIAS the search

Starting from St. Louis--Blind

BIASED SEARCH

Starting from St. Louis--Informed

Definition

reminder: Sigma = State space

A *heuristic function* (or simply heuristic) is a function $h: \Sigma \rightarrow \Re$, that takes a state as its argument and returns a real number that is an estimate of the distance (or cost) from that state to the closest (or having lowest-cost path) goal state.

$h(s) = r$

The function h is typically based on *partial information* about the relationship between each state s and the closest goal state γ to s.

For example, if each state has an (x,y) location, then knowing only x_s and x_γ, we could estimate the distance between s and g as $| x_s - x_\gamma |$.

typically, h(goal state) = 0

Outline

Motivation, Definition
A Algorithm*
Admissibility and Consistency
Heuristics for the 8 Puzzle
Designing Heuristics

A* Algorithm

variation on Dijstkrras algorithm that incorporates heuristic?

Given a state space Σ having a distance (or cost) function on moves (graph edges): $d(s_i, s_j)$, the A* algorithm searches for a shortest (lowest-cost) path from the initial state s_0 to a goal state γ.

The following algorithm gives the general control structure for A*. It omits a few details:

1. Back pointers for backtracing a path when a goal state is reached.
2. Details of computing g. (done in a manner similar to that in Dijkstra's algorithm, i.e., Uniform Cost Search).
3. Details of implementing the OPEN list and its methods for inserting, finding, and removing.

A* Algorithm

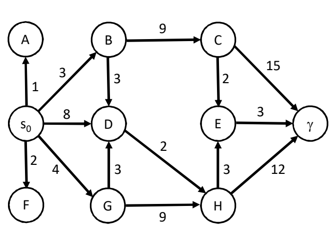
1. For the start state s_0, compute $f(s_0) = g(s_0)+h(s_0) = h(s_0)$ and put $[s_0,f(s_0)]$ on a list OPEN.
2. If OPEN is empty, output "DONE" and stop.
3. Find and remove the item [s,p] on OPEN having ~~lowest~~ **highest priorty** p. Break ties arbitrarily
   Put [s,p] on CLOSED.
   If s is a goal state: output its description (and backtrace a path), and
   if h is known to be admissible, halt.
4. Generate the list L of [s',f(s')] pairs where the s' are the successors
   of s and their f values are computed using f(s') = g(s')+h(s').
   Consider each [s',f(s')].
   - If there is already a pair [s', q] on CLOSED (for any value q):
     - if f(s') > q, then remove [s,f(s')] from L.
     - If f(s') ≤ q, then remove [s',q] from CLOSED.
   - Else if there is already a pair [s', q] on OPEN (for any value q):
     - if f(s') > q, then remove [s,f(s')] from L.
     - If f(s') ≤ q, then remove [s',q] from OPEN.
5. Insert all members of L onto OPEN.
6. Go to Step 2.

we found a shorter path to s' ;
must recompute all successors

A* Algorithm Behavior

During the search A* gives highest priority to that as-yet unexplored state (except in cases where some previously explored state needs to be re-examined) that has the lowest sum of distance from the initial state plus estimated distance to a goal.

Example

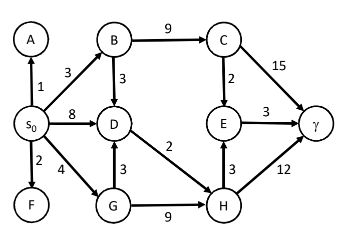


state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Example

OPEN: $[s_0,14]$



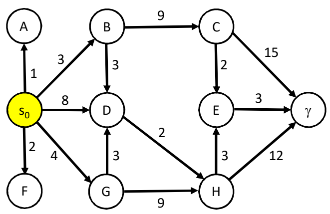
CLOSED:

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
[B,7], [D,11], [G,~~15~~ 14], [A,16], [F,18]

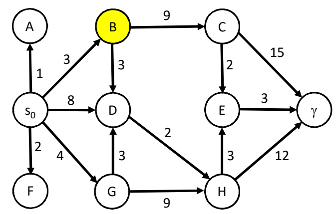
CLOSED: [s_0,14]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
~~[B,7], [D,11], [G,14], [A,16], [F,18]~~
[D,9], [G,15], [A,16], [F,18], [C,22]

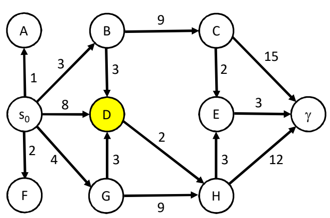
CLOSED: [s_0,14], [B,7]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
~~[B,7], [D,11], [G,14], [A,16], [F,18]~~
~~[D,9], [G,14], [A,16], [F,18], [C,22]~~
[H,13], [G,15], [A,16], [F,18], [C,22]

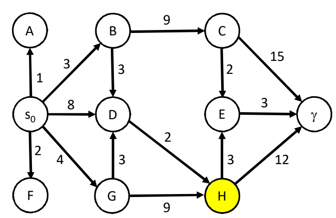
CLOSED: [s_0,14], [B,7], [D,9]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	5	2	16	8	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
~~[B,7], [D,11], [G,14], [A,16], [F,18]~~
~~[D,9], [G,14], [A,16], [F,18], [C,22]~~
~~[H,13], [G,15], [A,16], [F,18], [C,22]~~
[E,13], [G,15], [A,16], [F,18], [γ,20], [C,22]

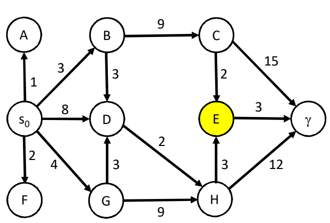
CLOSED: [s_0,14], [B,7], [D,9], [H,13]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
~~[B,7], [D,11], [G,14], [A,16], [F,18]~~
~~[D,9], [G,14], [A,16], [F,18], [C,22]~~
~~[H,13], [G,15], [A,16], [F,18], [C,22]~~
~~[E,13], [G,15], [A,16], [F,18], [γ,20], [C,22]~~
[γ,14], [G,15], [A,16], [F,18], [C,22]

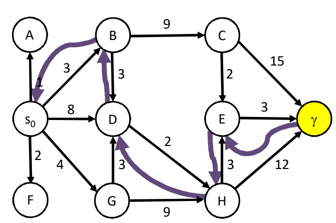
CLOSED: [s_0,14], [B,7], [D,9], [H,13], [E,13]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Ss State-space Search

Example



OPEN: ~~[s_0,14]~~
~~[B,7], [G,12], [D,13], [A,16], [F,18]~~
~~[D,9], [G,14], [A,16], [F,18], [C,22]~~
~~[H,13], [G,15], [A,16], [F,18], [C,22]~~
~~[E,13], [G,15], [A,16], [F,18], [γ,20], [C,22]~~
[G,15], [A,16], [F,18], [C,22]

Predecessor links that mark the shortest path

CLOSED: [s_0,14], [B,7], [D,9], [H,13], [E,13], [γ,14]

state s:	s_0	A	B	C	D	E	F	G	H	γ
heuristic h(s):	14	15	4	10	3	2	16	10	5	0

We show newly enqueued [s,p] pairs in green, and updated [s,p] pairs in red.

Notes on the Example

The path found is the same as that found by UCS (Dijkstra).
However, fewer nodes are expanded; e.g., A and F are never expanded.

The heuristic h is admissible, so as soon as γ becomes the current state, we can stop.

But h is not consistent: h(G)–h(D)>d(G,D).
This fact doesn't cause trouble here, fortunately.
(If h is consistent, no state will ever have to be expanded a second time, i.e., never have to be moved from CLOSED back to OPEN.)

Outline

Motivation, Definition
A* Algorithm
Admissibility and Consistency
Heuristics for the 8 Puzzle
Designing Heuristics

Note heuristic MUST be carefully chosen to ensure our path is actually minimal! The properties our heuristic must satisfy our Admissibility and Consistency

Admissibility and Consistency

Heuristic h is admissible if and only if
$$(\forall s \in \Sigma)(\forall \gamma \in \Gamma)\ h(s) \leq d(s, \gamma)$$

Here $d(s, \gamma)$ is the length (cost) of the shortest (lowest-cost) path from s to γ.

I.e., h estimates but never overestimates the distance from s to the closest goal.

If h is admissible, then A*, using h, finds a shortest path (optimal path) as soon as it expands any goal γ.

Admissibility. As soon as we first reach a goal state, we have reached it via a minimal path

Admissibility and Consistency

Heuristic h is *consistent* if and only if
For each edge (s_i, s_j) in the problem-space graph,
$$h(s_i) - h(s_j) \leq d(s_i, s_j)$$

Here $d(s_i, s_j)$ is the length (cost) of the edge from s_i to s_j.

If h is consistent, then along any shortest path from a node (state) s to its closest goal, then h values will be monotonically non-increasing along the path.

If h is consistent, then A*, using h, never has to re-expand a node. i.e. remove a node from CLOSED and re-expand it

Consistency: never have to visit and expand a state more than once i.e. ensures our search is efficient

Admissibility of A*

A* is *admissible*: Provided its heuristic is admissible, and a path exists from s_0 to γ, then A* will find a shortest path and stop.

Outline

Motivation, Definition
A* Algorithm
Admissibility and Consistency
Heuristics for the 8 Puzzle
Designing Heuristics

END LECT WED JAN 23

Heuristics for the Eight Puzzle

$h_0(s) = 0$ (uninformed; blind search)

$h_1(s)$ = number of tiles out of place. ("Hamming") $0 \le h_1(s) \le 8$.

$\gamma =$ (goal state)

	1	2
3	4	5
6	7	8

$s_a =$

3	1	2
7	6	5
4		8

$h_1(s_a)$ = 4.

h1 is admissable IF we ignore the blank space, otherwise its not (consider the final move; both blank space and one tile would be out of place)

h1 easy to compute

Heuristics for the Eight Puzzle

$h_2(s)$ = number of rows tile 7 is away from its place.

$\gamma =$

	1	2
3	4	5
6	7	8

$s_a =$

3	1	2
7	6	5
4		8

$h_2(s_a)$ = 1.

Using tile 7 is arbitrary here -- just an example of a heuristic based on a single, particular tile.

Heuristics for the Eight Puzzle

$h_3(s)$ = number of rows and columns tile 7 is away from its place.

$\gamma =$

	1	2
3	4	5
6	7	8

$s_a =$

3	1	2
7	6	5
4		8

$h_3(s_a)$ = 2.

This sum is known as the *Manhattan distance* (for a single tile).

Admissiable AND dominates h2; more informed = better
BUT harder to compute

Heuristics for the Eight Puzzle

$h_4(s)$ = sum of Manhattan distances for all 8 tiles.

$\gamma =$

	1	2
3	4	5
6	7	8

$s_a =$

3	1	2
7	6	5
4		8

$h_4(s_a)$ = 7.

This is called the Manhattan distance heuristic.
In this example $h_4(s_a) = h(s_a)$ (the actual shortest distance).

Admissable, but again hard to compute; we see this trade off between the ease to compute and how informed the heuristic is quite often

Heuristics for the Eight Puzzle

$h_5(s)$ = sum of Euclidean distances for all 8 tiles.
$= \sum_{i=1}^{8} \sqrt{dx_i^2 + dy_i^2}$

$\gamma =$

	1	2
3	4	5
6	7	8

$s_a =$

3	1	2
7	6	5
4		8

$h_5(s_a) = 1 + 3\sqrt{2} \approx 5.2326$

This is called the Euclidean distance heuristic.
In (at least) this example $h_5(s_a) < h_4(s_a)$.
Euclidean is not as good as Manhattan.

Heuristic Domination

If $(\forall s \in \Sigma)\ h_i(s) \ge h_j(s)$, then we say h_i dominates $h_j(s)$.
However, we assume both heuristics are admissible.

If h_i dominates h_j, then it we call h_i "more informed" than h_j.
Having a highly informed heuristic is good for limiting a search to relevant parts of the state space.

However, one has to trade off this off against the higher computational cost that usually goes with more informed heuristics.

Ss State-space Search

Outline

Motivation, Definition
A* Algorithm
Admissibility and Consistency
Heuristics for the 8 Puzzle
Designing Heuristics

Ss State-space Search

Designing Heuristics

A common approach to defining heuristics is to create a simpler type of problem. remove some of the constraints…

E.g., For the 8 Puzzle, allow tiles to be removed an put back anywhere, and "charge" a cost of 0.5 for each removal and 0.5 for each putting back. That leads to the Hamming heuristic.

Or: allow tiles to be piled up on top of one another, thus making it easier to move each tile (still one square at a time) to its destination. This leads to the Manhattan heuristic.

Ss State-space Search

Designing Heuristics (cont)

Another way to simplify: change some of the tiles into "blanks" (like the blank tile in Scrabble). The new goal is to get only the non-blank tiles into their proper positions; the blanks are "don't-care" tiles that still take up space, but whose relative ordering is not important. For example:

$\gamma' =$

	1	2
3	4	5
/	/	/

$s_a' =$

3	1	2
/	/	5
4		/

To compute $h(s_a')$, we transform s_a into s_a', and solve the simplified problem, getting a path length $d(s_a', \gamma')$, which we use as the value of h. If the reduced problem is easy enough, then we can precompute a table of $d(s_a', \gamma')$ values to speed up computing h during the search. Such a table is called a *pattern database*.

Ss State-space Search

Outline

Motivation, Definition
A* Algorithm
Admissibility and Consistency
Heuristics for the 8 Puzzle
Completeness and Optimality
Designing Heuristics
done!