# Informed Search
(or Heuristic Search)

Vincent A. Cicirello, Ph.D.
Professor of Computer Science
cicirelv@stockton.edu https://www.cicirello.org/

STOCKTON UNIVERSITY www.stockton.edu

1

Lesson 1

## INFORMED VS UNINFORMED SEARCH

STOCKTON UNIVERSITY www.stockton.edu

2

## Overview

- What is the difference between ***informed*** search and ***uninformed*** search?
- Heuristics: what are they? And what are they good for?
- Best-first greedy search
- A* Search
  - A* is complete
  - Admissible heuristics
  - A*'s dark side
- Iterative Deepening A* (IDA*) Search

STOCKTON UNIVERSITY www.stockton.edu

3

## Uninformed Search

- Uninformed search algorithms are search algorithms that don't rely on any problem-dependent knowledge.
  - They are **un**informed of the problem they are solving.
  - All of the search algorithms that we've seen so far: BFS, DFS, Least-Cost BFS, UCS, Depth-Limited DFS, Iterative Deepening.
- All of these have a worst-case runtime that is exponential in the average branching factor of the problem.
  - Not much we can do about that in general.
  - The problems of interest to A.I. are often NP-Complete or NP-Hard.

STOCKTON UNIVERSITY www.stockton.edu

4

## Informed Search

- Informed search algorithms use problem-dependent knowledge, in the form of a heuristic, for guidance.
  - They are informed of the problem they are solving.
  - A heuristic is used to attempt to more intelligently choose which successors appear to be more promising than the others.
- The informed search algorithms that we'll study also have worst-case runtime that is exponential in problem's average branching factor.
  - Not much we can do about that in general.
  - The problems of interest to A.I. are often NP-Complete or NP-Hard.
- With good heuristics, they tend to avoid exponentially-long runs.

STOCKTON UNIVERSITY www.stockton.edu

5

## Heuristics

Suppose in addition to the standard search specification we also have a *heuristic*.

*A heuristic function maps a state onto an estimate of the cost to the goal from that state.*

Can you think of examples of heuristics?

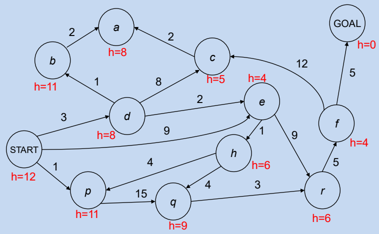
- E.G. for the 8-puzzle?
- E.G. for planning a path through a maze?

Denote the heuristic by a function $h(s)$ from states to a cost value.

STOCKTON UNIVERSITY www.stockton.edu

6

## Euclidian Heuristic




7

## Eight Puzzle Heuristic

- A few example heuristics for the 8-puzzle:
  - Number of tiles in the wrong spots
  - Sum of Manhattan distances
- Is one of these better than the other?
  - Yes: Sum of Manhattan distances

Example State

| | | |
|---|---|---|
| 1 | | 5 |
| 2 | 6 | 3 |
| 7 | 4 | 8 |

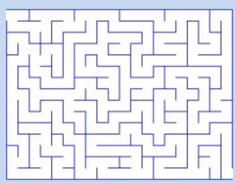
Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

8

## Maze Solving

- A few heuristics for maze solving:
  - Euclidian distance
  - Manhattan distance
- Is one of these better than the other?
  - Yes. Manhattan distance.



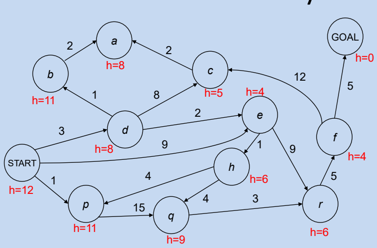
9

Lesson 2

## BEST-FIRST GREEDY SEARCH

10

## Best-First Greedy Search

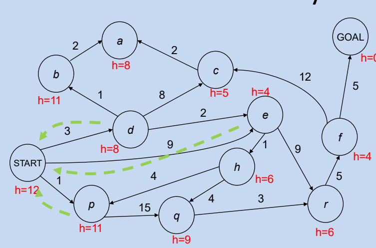



PQ = { (Start, 12) }

Priory-queue uses h() as priority.
Repeat:
1. Remove least priority
2. Add successors if not visited before, setting backpointers.
3. Terminate if goal found.

11

## Best-First Greedy Search

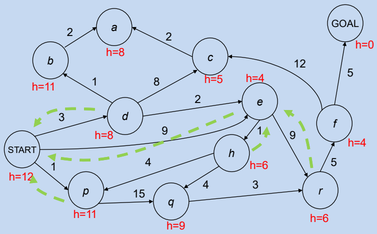



PQ = { (e, 4), (d, 8), (p, 11) }

Priory-queue uses h() as priority.
Repeat:
1. Remove least priority
2. Add successors if not visited before, setting backpointers.
3. Terminate if goal found.

12

## Best-First Greedy Search



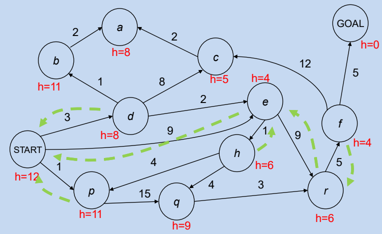

PQ = { (r, 6), (h, 6), (d, 8), (p, 11) }

Priory-queue uses h() as priority.
Repeat:
1. Remove least priority
2. Add successors if not visited before, setting backpointers.
3. Terminate if goal found.



13

## Best-First Greedy Search



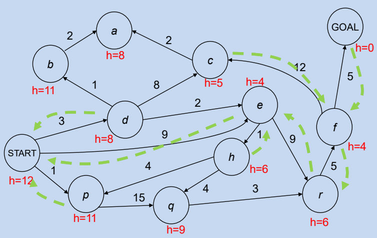

PQ = { (f, 4), (h, 6), (d, 8), (p, 11) }

Priory-queue uses h() as priority.
Repeat:
1. Remove least priority
2. Add successors if not visited before, setting backpointers.
3. Terminate if goal found.



14

## Best-First Greedy Search



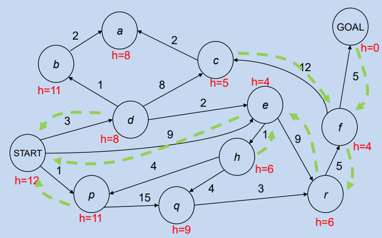

PQ = { (Goal, 0), (c, 5), (h, 6), (d, 8), (p, 11) }

Priory-queue uses h() as priority.
Repeat:
1. Remove least priority
2. Add successors if not visited before, setting backpointers.
3. Terminate if goal found.



15

## Best-First Greedy Search




PQ = { (c, 5), (h, 6), (d, 8), (p, 11) }

Goal is removed from PQ, so search returns the path implied by backpointers:

Start, e, r, f, Goal

With a cost of 9+9+5+5=28

Actual optimal path is:
Start, d, e, h, q, r, f, Goal
Cost: 3+2+1+4+3+5+5=23



16

## Best First "Greedy" Search

Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
    (s , h ) := Pop-least(PQ)
    foreach s' in succs(s)
    if s' is not already in PQ and s' never previously been visited
        Insert-PriQueue(PQ,s',h(s'))

| Algorithm | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| Best First Greedy Search | | | | |



17

## Best First "Greedy" Search

Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
    (s , h ) := Pop-least(PQ)
    foreach s' in succs(s)
    if s' is not already in PQ and s' never previously been visited
        Insert-PriQueue(PQ,s',h(s'))

| Algorithm | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| Best First Greedy Search | Y | | | |



18

## Best First “Greedy” Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
      (s , h ) := Pop-least(PQ)
      foreach s' in succs(s)
      if s' is not already in PQ and s' never previously been visited
            Insert-PriQueue(PQ,s',h(s'))
```

| Algorithm | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| Best First Greedy Search | Y | N | | |



19

## Best First “Greedy” Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
      (s , h ) := Pop-least(PQ)
      foreach s' in succs(s)
      if s' is not already in PQ and s' never previously been visited
            Insert-PriQueue(PQ,s',h(s'))
```

| Algorithm | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| Best First Greedy Search | Y | N | $O(log(Q)*min(BN,B^{LMAX}))$ | $O(min(N,B^{LMAX}))$ |



20

Lesson 3

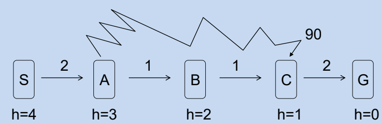
## A* SEARCH



21

## Let’s Make “Best first Greedy” Look Stupid!

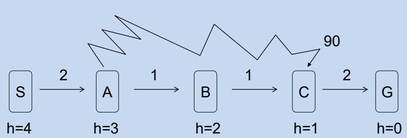
- PQ = {(S,4)}
- Expand S
  - PQ = {(A,3)}
- Expand A
  - PQ = {(C,1), (B,2)}
- Expand C
  - PQ = {(G,0), (B,2)}
- Expand G… found the goal… solution path S,A,C,G has cost 94…
- But a shorter path exists… S,A,B,C,G with cost 6






22

## Let’s Make “Best first Greedy” Look Stupid!




- Best-first greedy is clearly not guaranteed to find optimal
- Question: What can we do to avoid the stupid mistake?



23

## A* - The Basic Idea

- **Best-first greedy**: When you expand a node *s*, take each successor *s'* and place it on PriQueue with priority *h*(*s'*)
- **A***: When you expand a node *s*, take each successor *s'* and place it on PriQueue with priority

  (Cost of getting to *s'*) + *h*(*s'*) (1)

  Let *g*(*s*) = Cost of getting to *s* (2)

and then define…

$f(s) = g(s) + h(s)$ (3)

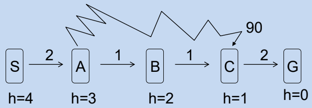


24

## A* Looking Non-Stupid

When we're at state A…
- We have 2 sucessors, B and C…
- $f(B) = g(B) + h(B) = 3 + 2 = 5$
- $f(C) = g(C) + h(C) = 92 + 1 = 93$
- PQ = { (B, 5), (C, 93) }

When we're at state B…
- One successor, C.
- $f(C) = g(C) + h(C) = 4 + 1 = 5$
- PQ = { (C, 5) }

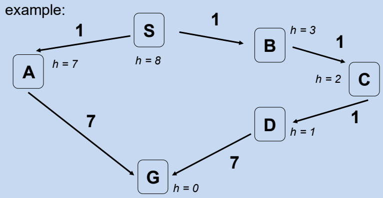



25

## When should A* terminate?

Idea: As soon as it generates a goal state?

Look at this example:

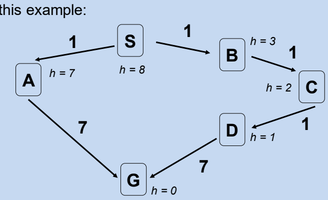



26

## When should A* terminate?

Idea: As soon as it generates a goal state?

Look at this example:

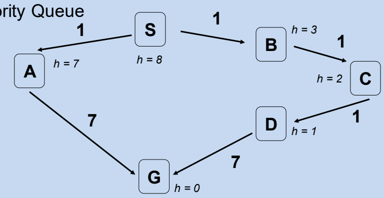



- Expand S
  - PQ = {(B,4),(A,8)}
- Expand B
  - PQ = {(C,4),(A,8)}
- Expand C
  - PQ = {(D,4),(A,8)}
- Expand D
  - PQ = {(A,8),(G,10)}
- A goal states has been generated

27

## Correct A* termination rule:

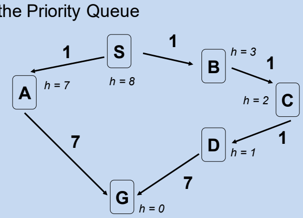
A* Terminates Only When a Goal State Is Removed from the Priority Queue




28

## Correct A* termination rule:

A* Terminates Only When a Goal State Is Removed from the Priority Queue

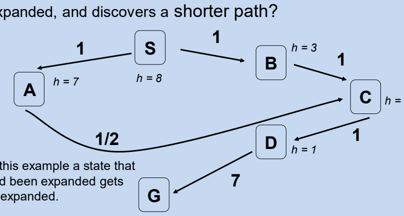



- Expand S
  - PQ = {(B,4),(A,8)}
- Expand B
  - PQ = {(C,4),(A,8)}
- Expand C
  - PQ = {(D,4),(A,8)}
- Expand D
  - PQ = {(A,8),(G,10)}
- Expand A
  - PQ = { (G,8) }
- Expand G
  - Found shorter path

29

## A* revisiting states

Another question: What if A* revisits a state that was already expanded, and discovers a shorter path?



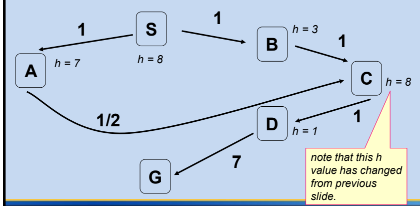

In this example a state that had been expanded gets re-expanded.

- Expand S
  - PQ = { (B,4), (A,8) }
- Expand B
  - PQ = { (C,4), (A,8) }
- Expand C
  - PQ = { (D,4), (A,8) }
- Expand D
  - PQ = { (A,8), (G,10) }
- Expand A
  - PQ = {(C,3.5), (G,10)}
  - Found less costly path to C.
- Expand C
  - PQ = {(D,3.5), (G,10)}
- Expand D
  - PQ = { (G, 9.5) }
- Expand G
  - Done
  - Path: S, A, C, D, G

30

## A* revisiting states

Another question: What if A* visits a state that is already on the queue?




- Expand S
  - PQ = { (B,4), (A,8) }
- Expand B
  - PQ = {(A,8), (C,10) }
- Expand A
  - PQ = { (C, 9.5) }
  - Found less costly path to C.
- Expand C
  - PQ = {(D,3.5)}
- Expand D
  - PQ = { (G, 9.5) }
- Expand G
  - Done
  - Path: S, A, C, D, G



31

## A* Search

Reminder: g(s) is cost of shortest known path to s

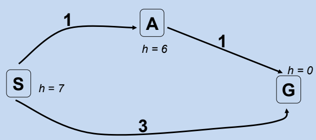
Reminder: h(s) is a heuristic estimate of cost to a goal from s

- Priority queue *PQ* begins empty.
- *V* (= set of previously visited (*state,f,backpointer*)-triples) begins empty.
- Put *S* into *PQ* and *V* with priority *f*(*S*) = *g*(*S*) + *h*(S)  = *h*(S) because *g*(start) = 0
- Is *PQ* empty?
  - Yes? There's no solution
  - No? Remove node with lowest *f*(*s*) from queue. Call it *s*.
  - If *s* is a goal, stop and report success.
  - "expand" *s* : For each s' in **successors**(s)….
    - Let *f'* = *g*(*s'*) + *h*(*s'*) = *g*(*s*) + *cost*(*s,s'*) + *h*(*s'*)
    - **If** *s'* not seen before, or *s'* previously expanded with *f*(*s'*)>*f'*, or *s'* currently in *PQ* with *f*(*s'*)>*f'*
    - **Then** Place/promote *s'* on priority queue with priority *f'* and update *V* to include (*state*=*s'*, *f'*, *BackPtr*=*s*).
    - **Else** Ignore *s'*



32

## Is A* Guaranteed to Find the Optimal Path?




- Expand S
  - PQ = { (G, 3), (A, 7) }
- Expand G
  - Done
  - Found path: S, G
  - Cost 3
- But a less costly path exists
  - S, A, G
  - Cost 2

Nope. And this example shows why not.

Next, we'll look at what we can do guarantee an optimal solution. We'll need to put requirements on the heuristic.



33

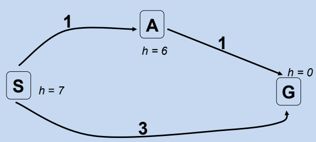
Lesson 4

## ADMISSIBLE HEURISTICS



34

## A* is not Optimal (at least not in general)




- Expand S
  - PQ = { (G, 3), (A, 7) }
- Expand G
  - Done
  - Found path: S, G
  - Cost 3
- But a less costly path exists
  - S, A, G
  - Cost 2

Nope. And this example shows why not.

**If we put some requirements on the heuristic, then we can get A* to find the optimal solution.**



35

## Admissible Heuristics

- Write $h^*(s)$ = the true minimal cost to goal from *s*.
- A heuristic h(*s*) is **admissible** if $h(s) <= h^*(s)$ for all states *s*.
- An admissible heuristic is guaranteed never to overestimate cost to goal.
- An admissible heuristic is optimistic.



36

## 8-Puzzle Example

Example State

| | | |
|---|---|---|
| 1 | | 5 |
| 2 | 6 | 3 |
| 7 | 4 | 8 |

Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

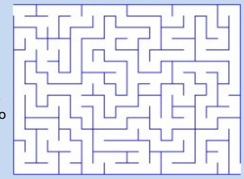
Which of the following are admissible heuristics?

- *h*(*s*) = Number of tiles in wrong position in state *s*
- *h*(*s*) = 0
- *h*(*s*) = Sum of Manhattan distances between each tile and its goal location
- *h*(*s*) = 1
- *h*(*s*) = min (2, *h**[*s*])
- *h*(*s*) = *h**(*n*)
- *h*(*s*) = max (2, *h**[*s*])

37

## Maze Solving

- A few **admissible** heuristics for maze solving:
  - Euclidian distance
  - Manhattan distance
- Both are admissible.
  - Without obstacles, shortest path between two points is a straight line (Euclidian distance), so will clearly never overestimate.
  - Only able to move left-right-up-down, so without walls, an optimal path would be to walk left-right straight line to x of goal, and then up-down to y of goal.



38

## A* and Optimality

- A* is complete.
  - Guaranteed to find a solution if one exists and to terminate otherwise.
- A* with an admissible heuristic is optimal.
  - Guaranteed to find the optimal path from the start to the goal.

39

Lesson 5

## DOMINATING HEURISTICS

40

## Comparing A* with Iterative Deepening

From Russell and Norvig

| | For 8-puzzle, average number of states expanded over 100 randomly chosen problems in which optimal path is length… | | |
|---|---|---|---|
| | …4 steps | …8 steps | …12 steps |
| Iterative Deepening (see slides on uninformed search) | 112 | 6,300 | $3.6 \times 10^6$ |
| A* search using "number of misplaced tiles" as the heuristic | 13 | 39 | 227 |
| A* using "Sum of Manhattan distances" as the heuristic | 12 | 25 | 73 |

41

## Comparing A* with It… …ng

Comments
1. At first sight might look like even "number of misplaced tiles" is a great heuristic. But probably h(state)=0 would also do much much better than ID, so the difference is mainly to do with ID's big problem of expanding the same state many times, not the use of a heuristic.
2. Judging solely by "number of states expanded" does not account for overhead of maintaining hash tables and priority queue for A*, though it's pretty clear here that this won't dramatically change the results.

There are only a couple hundred thousand states for the entire eight puzzle

| | … of …over 100 …en problems in …ath is length… | | |
|---|---|---|---|
| | …steps | …steps | …12 steps |
| | | …00 | $3.6 \times 10^6$ |
| A* search using "number of misplaced tiles" as the heuristic | … | 39 | 227 |
| A* using "Sum of Manhattan distances" as the heuristic | 12 | 25 | 73 |

42

## Dominating Heuristics

- Given 2 admissible heuristics for a problem, one might ask whether one is always better?
- **Example:** *Is Manhattan distance always better than number of misplaced tiles for the 8-puzzle?* Yes
- A heuristic $h_2$ **dominates** $h_1$ if for all states s, $h_2(s) >= h_1(s)$
- Using A* with heuristic $h_2$ will never expand more nodes than A* with $h_1$.
- It is always better to use a heuristic function with higher values, provided:
  - it is admissible (never overestimates)
  - and that the computation time for the heuristic is not too large

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

43

## Additional Examples of Dominating Heuristics

- For the eight puzzle (and sliding tile puzzles in general), Manhattan distance dominates number of misplaced tiles.
- What about maze solving (assuming the maze is a grid with horizonal and vertical movement only)?
- Manhattan distance dominates Euclidian distance.
  - Euclidean distance (single straight line cutting through walls, etc) is shortest path if no obstacles and if free to move in any direction
  - Manhattan distance (two straight lines, cutting through walls, etc, one horizontally, and one vertically)

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

44

Lesson 6

## ITERATIVE DEEPENING A* SEARCH (IDA*)

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

45

## A* : The Dark Side

- A* can use lots of memory. In principle:

  *O(number of states)*
- For really big search spaces, A* will run out of memory.

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

46

## IDA* : Memory Bounded Search

- Iterative deepening A*. Actually, pretty different from A*. Assume costs integer.
  1. Do path-checking DFS, not expanding any state with *f*(*s*) > 0. Did we find a goal? If so, stop.
  2. Do path-checking DFS, not expanding any state with *f*(*s*) > 1. Did we find a goal? If so, stop.
  3. Do path-checking DFS, not expanding any state with *f*(*s*) > 2. Did we find a goal? If so, stop.
  4. Do path-checking DFS, not expanding any state with *f*(*s*) > 3. Did we find a goal? If so, stop.

  …keep doing this, increasing the *f*(*s*) threshold by 1 each time, until we stop.

  …need to also take care that we detect if problem not solvable (e.g., check if we cut-off search with limit)
- This is
  - Complete
  - Guaranteed to find optimal (if heuristic is admissible)
  - More costly than A* in general.
  - **But only requires memory that is linear in path length.**

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

47