# Heuristic Search



Daniel D. Harabor
Monash University



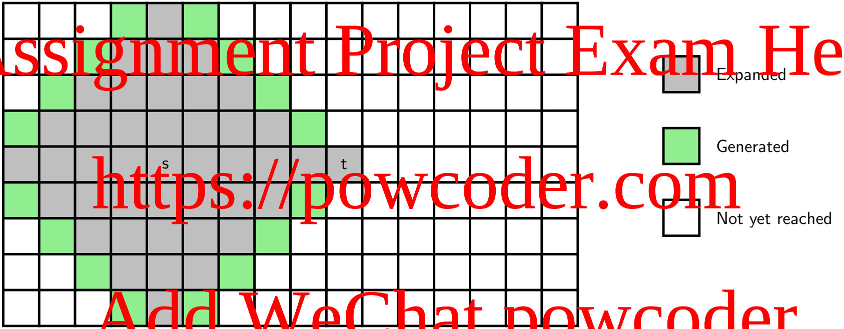
FIT3080

# Recap

`Uniform-Cost Search` is awesome, right?

- Complete
- Optimal
- No duplicates (`Graph-Search` version)
- Best-first expansion order
- Usually much faster than depth-first or breadth-first

# The problem with Uniform-Cost Search

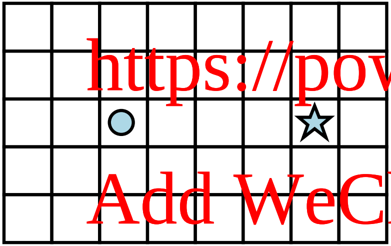





UCS has no idea where the target is. It's searching **blindly**.

We say an algorithm is **informed** if it relies on problem-specific knowledge that helps us decide which node to expand next.









One approach for making a search informed is to add a **heuristic function** — $h(n)$ — that estimates cost-to-go: from any node $n$ to the current goal.

We say an algorithm is **informed** if it relies on problem-specific knowledge that helps us decide which node to expand next.



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



One approach for making a search informed is to add a **heuristic function** — $h(n)$ — that estimates cost-to-go: from any node $n$ to the current goal.

# Heuristics, concretely

A heuristic is a function that satisfies the following **essential** properties:

- $h(n)$ is an estimate of the true path cost, from $n$ to $t$
- $h(n) \geq 0$
- $h(t) = 0$

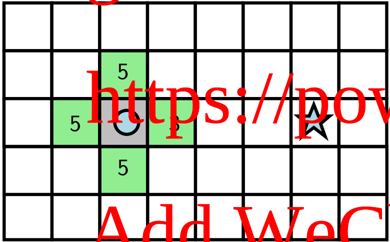


Where do heuristics come from?

- Human knowledge of the domain
- From solving a **relaxed** version of the problem at hand.
- From prior experience

**Idea:** the heuristic value $h(n)$ is the priority of each node.







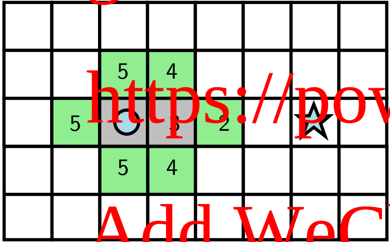



Here we use a heuristic called **Manhattan distance**:

$h_M(n) = \Delta x + \Delta y$

**Idea:** the heuristic value $h(n)$ is the priority of each node.



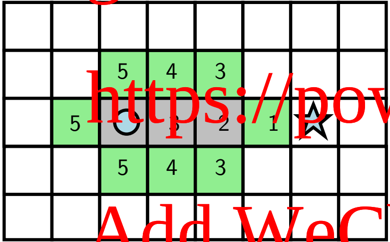





At each step, we expand the most promising node according to $h_M$.

**Idea:** the heuristic value $h(n)$ is the priority of each node.



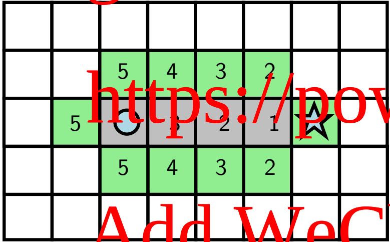





At each step, we expand the most promising node according to $h_M$.

**Idea:** the heuristic value $h(n)$ is the priority of each node.







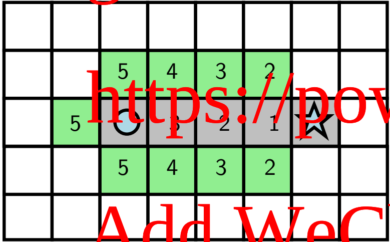



At each step, we expand the most promising node according to $h_M$.

# Greedy Best-First Search

**Idea:** the heuristic value $h(n)$ is the priority of each node.



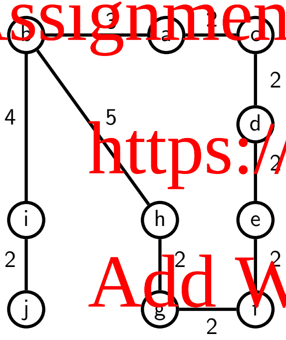





For this problem, $h_M$ is a perfect guide and the search cost is minimal.

In this example we seek an optimal path from **a** to **h**.








| Node | $h_{SLD}(n)$ |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |
| j | 3.6 |

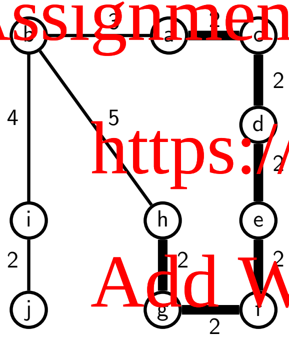


This time our heuristic is **straight line distance**
$h_{SLD} = \sqrt{\Delta x^2 + \Delta y^2}$

# The Problem with Greedy Best-First Search

In this example we seek an optimal path from **a** to **h**.






| Node | $h_{SLD}(n)$ |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |
| j | 3.6 |

Now GBFS becomes misleading and produces suboptimal results.
In `Tree-Search`, expanding greedily may not even terminate!

# The A* Algorithm

**Idea:** let's combine cost-so-far with cost-to-go. Nodes are now expanded according to minimum **f-value** where $f(n) = g(n) + h(n)$



Each $f(n)$ is an **estimate** of the total solution cost: from $s$ to $t$ via node $n$.



$$s \xrightarrow{g(n)} n \overset{h(n)}{\dashrightarrow} t$$

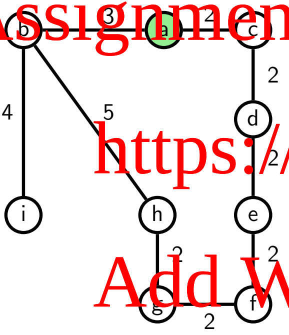


**NB:** UCS and GBFS are both special cases of A*.

A* originally appears in **(Hart, P.E., Nilsson, N.J. and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), pp.100-107.)**

In this example we seek an optimal path from **a** to **h**.



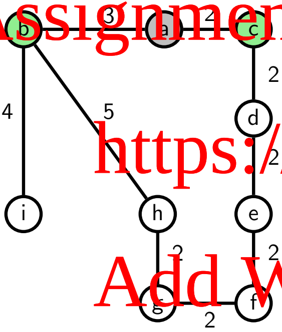



| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |

EXPANDING:
OPEN: [ (**a**, 4) ]

# Searching with A*

In this example we seek an optimal path from **a** to **h**.



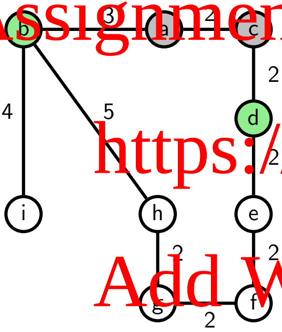

| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |

EXPANDING: (**a**, 4)
OPEN: [ (**c**, 6.5), (**b**, 8) ]

# Searching with A*

In this example we seek an optimal path from **a** to **h**.



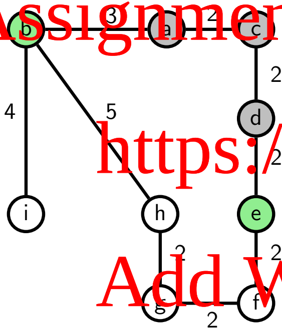



| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |

EXPANDING: (**c**, 6.5)
OPEN: [ (**d**, 6.8), (**b**, 8) ]

In this example we seek an optimal path from **a** to **h**.








| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| [illegible] | 3 |

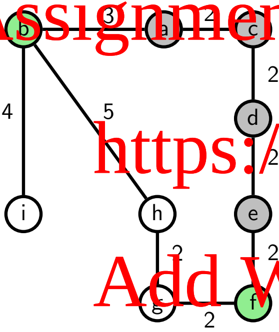


EXPANDING: (**d**, 6.8)
OPEN: [ (**e**, 8), (**b**, 8) ]

# Searching with A*

In this example we seek an optimal path from **a** to **h**.



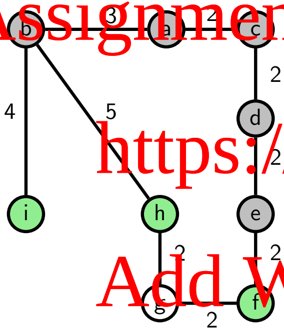



| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |

EXPANDING: (**e**, 8)
OPEN: [ (**b**, 8), (**f**, 10.8) ]

In this example we seek an optimal path from **a** to **h**.



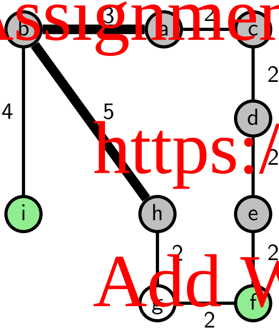



| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |

EXPANDING: (**b**, 8)
OPEN: [ (**h**, 8), (**i**, 10), (**f**, 10.8) ]

In this example we seek an optimal path from **a** to **h**.








| Node | $h_{SLD}$ estimate |
|---|---|
| a | 4 |
| b | 5 |
| c | 4.5 |
| d | 2.8 |
| e | 2 |
| f | 2.8 |
| g | 2 |
| h | 0 |
| i | 3 |



EXPANDING: (**h**, 8)
OPEN: [ (**i**, 10), (**f**, 10.8) ]

## Demo Time

Let's compare Uniform Cost Search (aka. **Dijkstra's algorithm**) with A*.

https://www.movingai.com/SAS/ASM/



- Here we solve pathfinding problems on **8-connected** gridmaps.
- Diagonal moves are allowed.
- The heuristic is **octile distance**

  $h_{oct} = \arg\min(\Delta x, \Delta y) \times \sqrt{2} + \arg\max(\Delta x, \Delta y) - \arg\min(\Delta x, \Delta y)$

# IDA*

The `Tree-Search` version of A* can be effectively combined with iterative deepening. Similar to DFID but now we search with $f$-costs.



IDA* sketch:

1. Depth-first tree-search with a **cost limit** (initially $f_{lim} = h(start)$)
2. Compute for each generated node an $f$-value estimate: $f(n) = g + h$
3. Expand all nodes with $f(n) \leq f_{lim}$
4. Update $f_{lim}$ to min. $f$-value of any node generated but not expanded.
5. Repeat 1-4 until the goal is found or until the tree is exhausted.

# Demo Time

Let's see how IDA* works

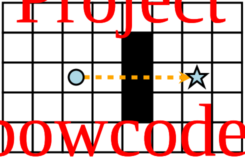
https://www.movingai.com/SAS/IDA/



- Here we solve a 3x2 sliding tile puzzle.
- The heuristic is **sum of Manhattan distances**.
- NB: The demo will show the entire tree, then the traversal.

**Admissibility:** the heuristic never over-estimates the cost-to-go:





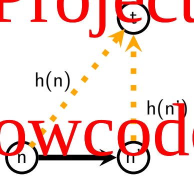


$h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from $n$ to $t$.

# Two important heuristic properties

**Consistency:** heuristic estimates decrease monotonically from $n$ to $t$.








$$h(n) \leq c(n, n') + h(n') \mid n' \in \textsc{Successors}(n)$$



Any consistent heuristic is also admissible. This includes $h_{SLD}$ and $h_M$.

## Optimality of A* (Graph-Search)

**Theorem**

`Graph-Search` A* is complete and optimal if its heuristic is **consistent**.



**Proof.**

(Sketch)

- The OPEN list covers every path to the target



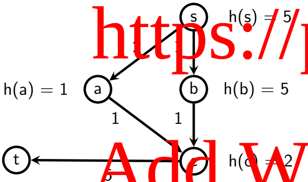
- Every graph node, including the goal, is eventually expanded
- Along any path $f$-values are non-decreasing (due to consistency)
- When A* selects a node $n$ for expansion $g(n)$ is the optimal path cost, from $s$ to $n$.



□

For any node $n$ and successor $n'$: we say the heuristic $h$ is **inconsistent** if

$$|h(n) - h(n')| \geq c(n, n')$$



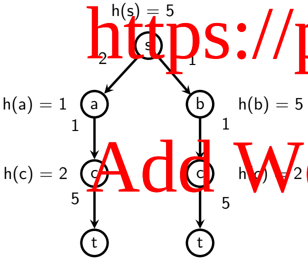

Inconsistent heuristics lead to re-expansions for optimal A* in graphs.

The news is not all bad and inconsistent heuristics are often useful in practice. See (**Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., & Zhang, Z. (2011). Inconsistent heuristics in theory and practice. Artificial Intelligence, 175(9-10)**)

For any node $n$ and successor $n'$: we say the heuristic $h$ is **inconsistent** if

$$|h(n) - h(n')| > c(n, n')$$








In trees, A* search generates all paths to each node. Here **admissibility** is sufficient to guarantee optimality.



The news is not all bad and inconsistent heuristics are often useful in practice. See (**Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., & Zhang, Z. (2011). Inconsistent heuristics in theory and practice. Artificial Intelligence, 175(9-10)**)

# Informedness

Heuristic $h_1$ is **less informed** than heuristic $h_2$ iff for all non-goal nodes $n$ we have $h_1(n) < h_2(n)$.

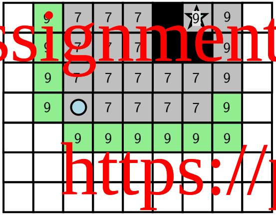


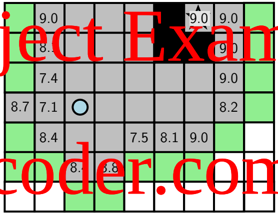
Heuristic $h_2$ **dominates** heuristic $h_1$ iff for all non-goal nodes $n$ we have $h_2(n) \geq h_1(n)$.



- More informed heuristics are better (less expansions, faster search).
- The value $|h(n) - h^*(n)| \geq 0$ is called the **heuristic error**.

# Informedness (2)





On 4-connected gridmaps $h_M$ dominates $h_{SLD}$.

# So, how good is A* really?

A* is **optimally efficient**. This means there exists no algorithm $A$ which is less informed than A* and that can possibly expand fewer nodes



Suppose there exists an algorithm $A_{h_1}$ that is is more clever than $A^*_{h_2}$ despite $h_1$ being less informed than $h_2$. That means there exists some node $n$ where:



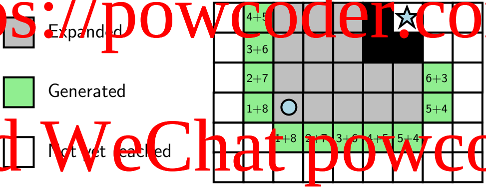
- $f_{h_1}(n) \geq f_{h_2}(n)$ ($A_{h_1}$ doesn't expand $n$)
- Implies $h_1(n) \geq h_2(n)$
- But $h_1$ is less informed than $h_2$ (contradiction)



The optimal efficiency of A* holds **up to tie-breaking**

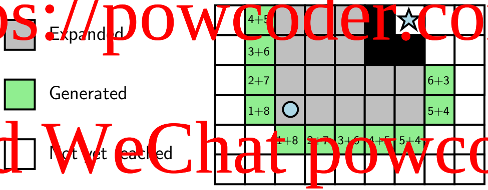
Sometimes many nodes have the same $f$-value. How to decide which node to expand next?





Sometimes many nodes have the same $f$-value. How to decide which node to expand next?



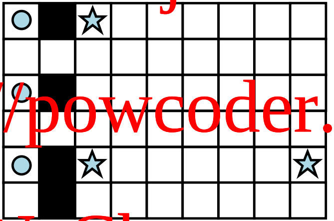



One good strategy: choose the node with smallest **h-value** (= largest-g)

# Tie Breaking with zero-cost actions

The smallest-h strategy fails when we allow **zero cost** actions.





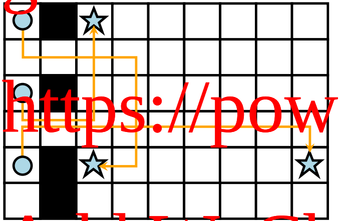


In this problem we assign paths to agents and need to cover every target. The objective is **makespan**: minimise the task completion time.

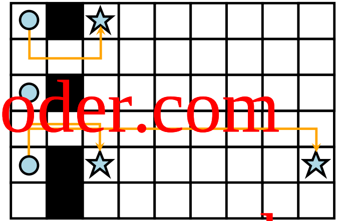
# Tie Breaking with zero-cost actions

The smallest-h strategy fails when we allow **zero cost** actions.







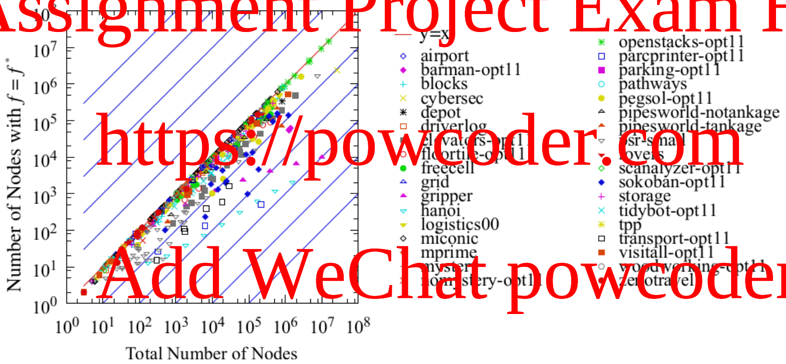




These two solutions have the same cost (and h-value) under makespan.
Other ideas: $[h, \mathit{lifo}]$, $[h, \mathit{fifo}]$, $[h, \mathit{rand}]$, $[\ldots]$. **None are dominant**.

How important is tie-breaking, really? Some results on IPC benchmarks.



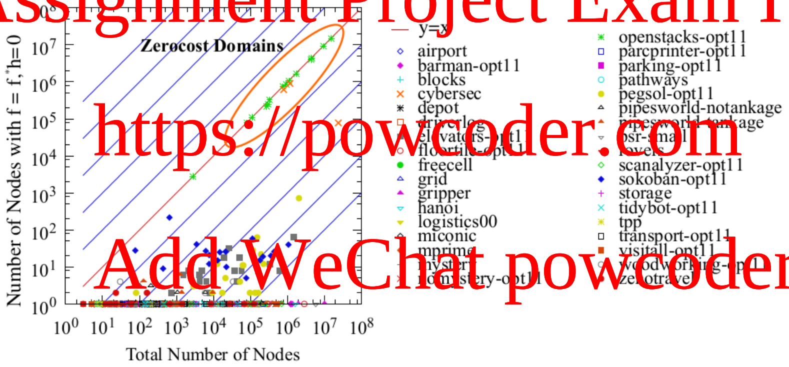



From the paper (**Asai, Masataro, and Alex Fukunaga. "Tie-breaking strategies for cost-optimal best first search." Journal of Artificial Intelligence Research 58 (2017)**)

# Tie Breaking (3)

How important is tie-breaking, really? Some results on IPC benchmarks.










From the paper (**Asai, Masataro, and Alex Fukunaga. "Tie-breaking strategies for cost-optimal best first search." Journal of Artificial Intelligence Research 58 (2017)**)

# A* performance characteristics

**In general:**

- For any dominant heuristic $h$ there exists a class of A* algorithms $\mathbf{A}^*_h$.
- Each $A^*_h \in \mathbf{A}^*_h$ is differentiated from the rest by tie-breaking.
- How to choose the one that always gives optimal efficiency in general is an **open problem**.

**Bottom line:**

- A* has the same worst-case performance as UCS.
- With a good heuristic, A* can be much more efficient **in practice**.
- Many of the currently leading planners employ some type of A* search.

## Improvements for A*

A variety of approaches exist for improving the performance of optimal A*.



- Better data structures for maintaining OPEN and CLOSED
- More accurate heuristics.



- Reducing the size of the search space via abstraction
- Constraint-based pruning (feasibility cuts, symmetry cuts etc)

A variety of approaches exist for improving the performance of optimal A*.



- Better data structures for maintaining OPEN and CLOSED
- More accurate heuristics.



- Reducing the size of the search space via abstraction
- Constraint-based pruning (feasibility cuts, symmetry cuts etc)

**or**



We can just give up on optimality.

# Relaxing the optimality criterion

We don't always need optimal solutions. Sometimes a **near optimal** or even a **feasible** solution is enough.



Types of suboptimality guarantees:



- Incomplete, unbounded (very fast, might fail)
- Complete, unbounded (very fast, variable solution quality)
- Bounded suboptimal (fast, solutions have quality guarantees)
  - **Relative suboptimality** (guarantee: $C \leq w \times C^*$)
  - Additive suboptimality (guarantee: $C \leq \delta + C^*$)

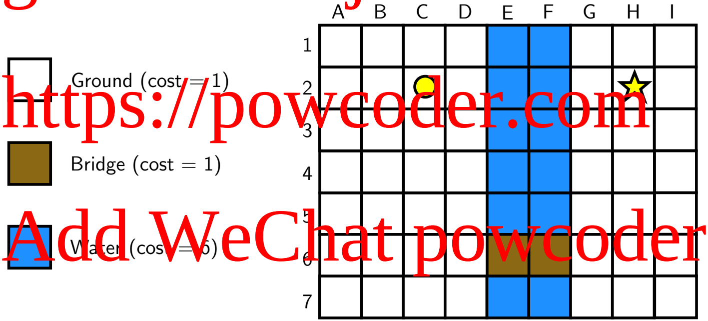


In this lecture we only discuss relative suboptimality. But additive suboptimality algorithms are interesting and useful! See: **(Valenzano, R.A., Arfaee, S.J., Thayer, J., Stern, R. and Sturtevant, N.R., "Using alternative suboptimality bounds in heuristic search." Twenty-Third International Conference on Automated Planning and Scheduling. 2013.)**

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.





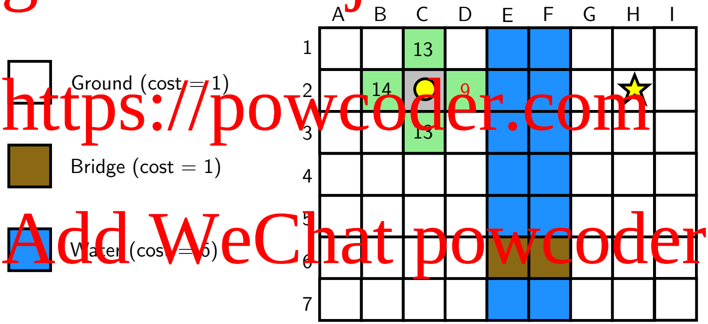





Multi-terrain gridmap (4c). Our estimator is $w \times h_M$ with w $= 2$.

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.



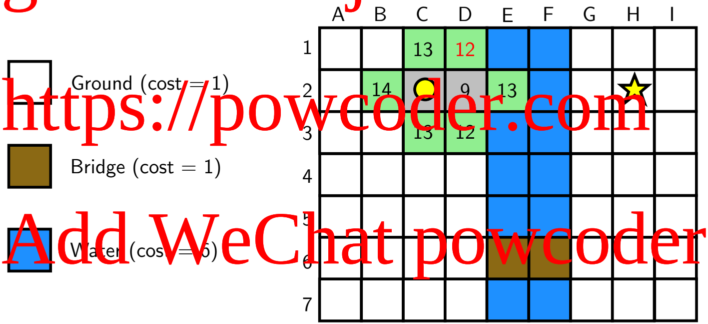





Minimum f-value = 9

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.







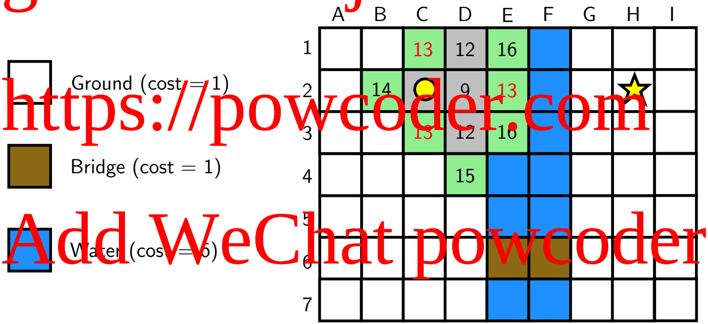



Minimum f-value = 12

# Weighted A*

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.



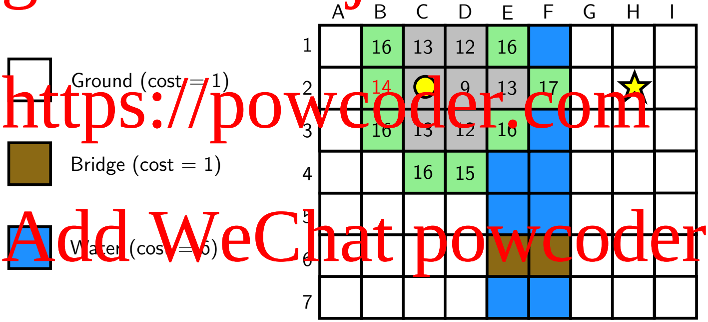





Minimum f-value = 13

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.







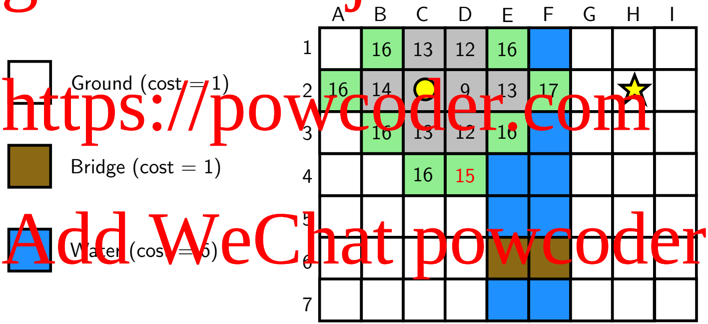



Minimum f-value = 14

# Weighted A*

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.







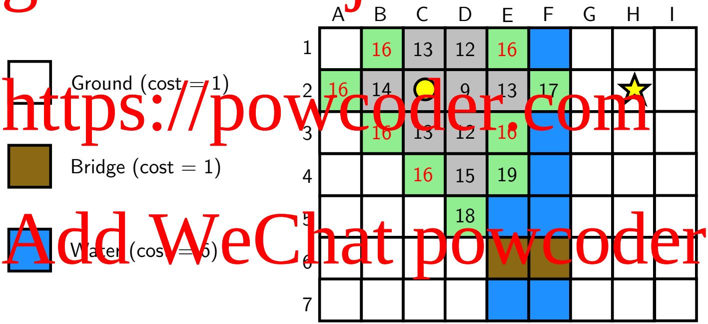



Minimum f-value = 15

# Weighted A*

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.







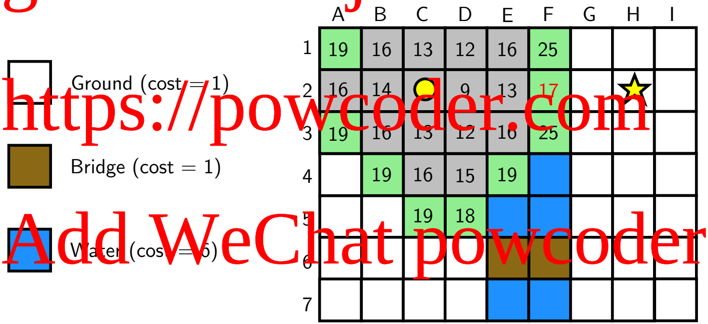



Minimum f-value = 16

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.





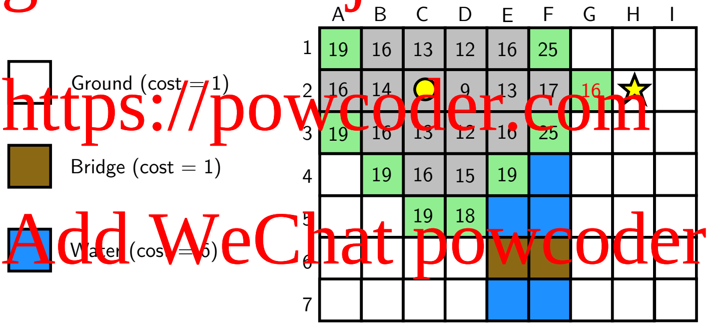





Minimum f-value = 17

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.





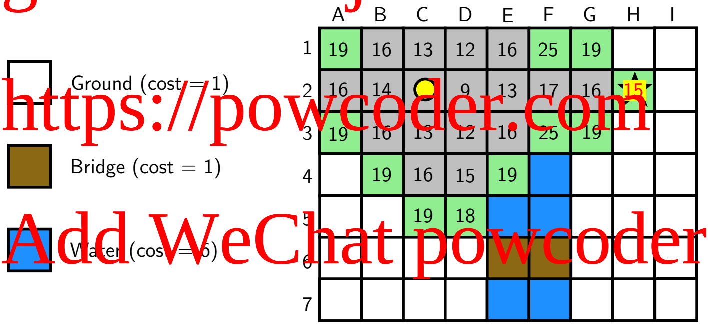





Minimum f-value = 16

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.





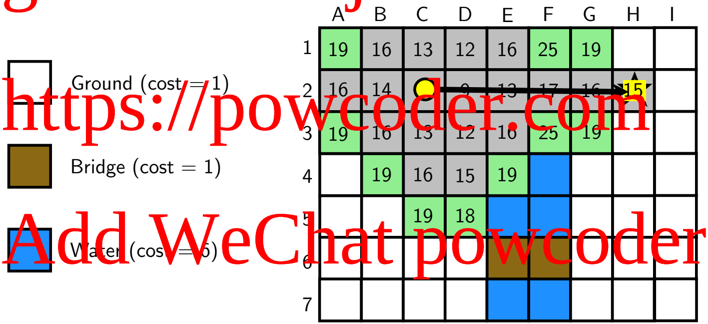





Minimum f-value = 15

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.







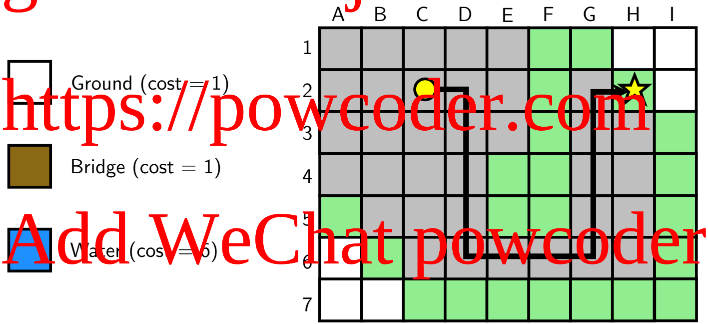



WA* solution (cost = 15)

# Weighted A*

**Idea:** Multiply A* heuristic estimates by some factor $w \geq 1$.








A* solution (cost = 13)

# Pros and Cons of WA*



The bad news:

- Heuristic estimates might **overestimate** the cost-to-go (inadmissible).
- Heuristic estimates might also be **inconsistent**.



The good news:

- Search now progresses more quickly toward the goal.
- We obtain feasible solutions sooner.
- Solution cost guaranteed: **at most** $w$-times larger than optimal.
- Re-expansions are **not required** to achieve the guarantee.

# Bounded guarantee of Weighted A*



**Theorem**

Let $f(n) = g(n) + w \times h(n)$ be the estimate for any generated node, with $h$ admissible and $w \geq 1$. Then, any returned solution has a cost $C \leq w \times C^*$



*Proof.*

**Base case ($n = s$):** trivially true.

**Inductive case:** On termination the goal node $G$ may still have an optimal ancester $n$ on the OPEN list. We have:



1. $f(G) \leq f(n)$ **(since $G$ was expanded first)**
2. $f(n) = g(n) + w \times h(n)$ **(by definition)**
3. $g(n) + w \times h(n) \leq w \times (g(n) + h(n))$ **(because algebra)**
4. $w \times (g(n) + h(n)) \leq w \times C^*$ **(since $f(n)$ is an underestimate)**

$\square$

# Variants of Weighted A*

Weighted A* is actually a family of related algorithms.



- $w = 0 \rightarrow$ Uniform Cost Search
- $w = 1 \rightarrow$ A*
- $w = \infty \rightarrow$ Greedy Best-First Search



More generally, WA* distributes suboptimality between $g$ and $h$ as follows:

$$f(n) = (1-\epsilon)g(n) + \epsilon h(n) \mid 0 \leq \epsilon \leq 1$$



Equivalent to multiplying the heuristic estimate by $w$ when $\epsilon = \frac{w}{1+w}$

Researchers have also tried dynamically adjusting the g- and h- value weights during search. See **(Köll, A.L. and Kaindl, H. "A new approach to dynamic weighting". In ECAI. 1992. (pp. 16-17))** (sadly not online; ask a librarian).

Let's compare A* and Weighted A*

https://www.movingai.com/SAS/ASM/



- Here we again solve pathfinding problems on **8-connected** gridmaps.
- Diagonal moves are allowed.
- The heuristic is **octile distance**

# Anytime Weighted A*

Weighted A* terminates after finding the first solution. But the search could also **continue** to optimality or until some time limit.



Sketch:

- Upper-bound (UB): the cost of the best solution thus far.
- Lower-bound (LB): unweighted f-value of the best node on OPEN.
- Keep expanding while LB $<$ UB.
- **Re-expand** nodes if their g-value can be improved.
- Update UB every time the goal is expanded anew.
- Terminate out of time or when OPEN is exhausted.



AWA* is a fascinating algorithm and the original paper is very accessible. See here: **(Hansen, E.A. and Zhou, R., 2007. Anytime heuristic search. Journal of Artificial Intelligence Research, 28, pp.267-297.)**

# EOF

## Next week

- Adversarial Search



## Administrivia

- Assignment 1 out soon
- Participate on the Ed forum
- Attend the tutorials!