Search

Search

State space graph.
Search


State space graph.

Given a task, model of world appropriate to task.
Search


State space graph.

Given a task, model of world appropriate to task.

Search Tree.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
    (also paths through state space graph.)

State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
    (also paths through state space graph.)

Uninformed Search.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
  (also paths through state space graph.)

Uninformed Search.
  Depth First Search.

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
  (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.

State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
  (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.
    Bad space bound, finds plan with fewest number of actions.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
    (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.
    Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
    (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.
    Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.
  Bad space bounds, finds optimal plan.
Search


State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
    (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.
    Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.
  Bad space bounds, finds optimal plan.

Demo: L3D1
Informed Search
Search Heuristics

A heuristic is:

▶ A function that estimates how close a state is to a goal
▶ Designed for a particular search problem
▶ Examples:
  - Euclidean distance for pathing.
  - Manhattan distance.
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples:
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Euclidean distance for pathing.
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Euclidean distance for pathing. Manhattan distance.
Example: Heuristic Function
Example: Heuristic Function

Heuristic:

The number of the largest pancake that is still out of place
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place
Greedy Search
Greedy Search

Expand the node that seems closest
Greedy Search

Expand the node that seems closest to the goal?

Better: 140 + 80 + 97 + 101 = 418
Greedy Search

Expand the node that seems closest to the goal?

What can go wrong?
Greedy Search

Expand the node that seems closest to the goal?

What can go wrong?

Greedy: 140 + 99 + 211 =
Greedy Search

Expand the node that seems closest to the goal?

What can go wrong?
Greedy: $140 + 99 + 211 = 450$
Better: $140 + 80 + 97 + 101 =$
Greedy Search

Expand the node that seems closest to the goal?

What can go wrong?

Greedy: 140 + 99 + 211 = 450
Better: 140+80+97+101 = 418
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Perfect heuristic.
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Perfect heuristic. A common case:
Best-first takes you straight to the (wrong) goal
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Perfect heuristic. A common case:
Best-first takes you straight to the (wrong) goal
Worst-case:
Like a badly-guided DFS
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Perfect heuristic. A common case:
Best-first takes you straight to the (wrong) goal

Worst-case:
Like a badly-guided DFS

[Demo contours greedy empty (L3D3)]
[Demo: contours greedy pacman small maze (L3D4)]
Video of Demo Contours Greedy (Empty)
Video of Demo Contours Greedy (pacman/small)
A* Search
A* Search

UCS

Greedy
A* Search

UCS

Greedy

A*
Combining UCS and Greedy

Uniform-cost orders by path cost, or backward cost: \( g(n) \).
Greedy orders by goal proximity, or forward cost: \( h(n) \).
A* Search orders by the sum: \( f(n) = g(n) + h(n) \).

Blue \( g(n) \), Red \( h(n) \).
Combining UCS and Greedy

**Uniform-cost** orders by path cost, or backward cost: \( g(n) \).
**Greedy** orders by goal proximity, or forward cost: \( h(n) \).
**A*** Search orders by the sum: \( f(n) = g(n) + h(n) \).

Blue \( g(n) \), Red \( h(n) \).
Combining UCS and Greedy

**Uniform-cost** orders by path cost, or backward cost: \( g(n) \).

**Greedy** orders by goal proximity, or forward cost: \( h(n) \).

\( A^* \) Search orders by the sum: \( f(n) = g(n) + h(n) \).

Blue \( g(n) \), Red \( h(n) \).

Uniform Cost Search.
Combining UCS and Greedy

**Uniform-cost** orders by path cost, or backward cost: $g(n)$.

**Greedy** orders by goal proximity, or forward cost: $h(n)$.

**A* Search** orders by the sum: $f(n) = g(n) + h(n)$.

Blue $g(n)$, Red $h(n)$.
Combining UCS and Greedy

**Uniform-cost** orders by path cost, or backward cost: \( g(n) \).

**Greedy** orders by goal proximity, or forward cost: \( h(n) \).

**A* Search** orders by the sum: \( f(n) = g(n) + h(n) \).
When should A* terminate?

Should we stop when we enqueue a goal?

\[ h = 2 \]

\[ h = 3 \]

\[ h = 2 \]

\[ h = 0 \]

\[ h = 1 \]

\[ h = 1 \]
When should A* terminate?

Should we stop when we enqueue a goal?

$h = 2$

$G$ in queue when $B$ expanded.

No: only stop when we dequeue a goal.
When should A* terminate?

Should we stop when we enqueue a goal?

\[ h = 2 \]

\[ h = 3 \]

\[ h = 1 \]

\[ h = 0 \]
When should A* terminate?

Should we stop when we enqueue a goal?

$S$ in queue when $B$ expanded.
When should A* terminate?

Should we stop when we enqueue a goal?

$h = 2$

G in queue when B expanded.
No: only stop when we dequeue a goal.
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost

We need estimates to be less than actual costs!

Lower bounds for actual costs.
Is A* Optimal?

What goes wrong?

- Actual goal cost < estimated goal cost
- We need estimates to be less than actual costs!

Lower bounds for actual costs.
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost
We need estimates to be less than actual costs!
Lower bounds for actual costs.
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost

We need estimates to be less than actual costs!
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost

We need estimates to be less than actual costs!
Lower bounds for actual costs.
Admissible Heuristics
Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe.
Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe.

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs.
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.
Admissible Heuristics

A heuristic $h$ is **admissible** (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

*Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.*
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice. Examples:
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
- Largest out of place pancake.

Admissible? No. For number of flips. Yes. For height of stack.

- Manhattan distance.
- Euclidean Distance.

Admissible? Yes!
Admissible Heuristics

A heuristic $h$ is **admissible** (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. 

**Note:** Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
  - Largest out of place pancake.
    - Admissible? No. For number of flips.
    - Yes. For height of stack.
  - Manhattan distance.
    - Admissible? Yes!
  - Euclidean Distance.
    - Admissible? Yes.
Admissible Heuristics

A heuristic $h$ is **admissible** (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
- Largest out of place pancake.
Admissible Heuristics

A heuristic $h$ is **admissible** (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. **Note:** Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
- Largest out of place pancake.
- Manhattan distance.
- Euclidean Distance.
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
- Largest out of place pancake. Admissible?
Admissible Heuristics

A heuristic \( h \) is admissible (optimistic) if:

\[
0 \leq h(n) \leq h^*(n)
\]

where \( h^*(n) \) is the true cost to a nearest goal

Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

Optimize: number of flips.

Largest out of place pancake.
Admissible?
  No. For number of flips.
  Yes. For height of stack.
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
- Largest out of place pancake. Admissible?
  - No. For number of flips.
  - Yes. For height of stack.
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
  - Largest out of place pancake.
  - Admissible? No. For number of flips.
  - Yes. For height of stack.

- Manhattan distance.
Admissible Heuristics

A heuristic $h$ is **admissible** (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. 

*Note:* Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.  
  - Admissible?
  - No. For number of flips.  
  - Yes. For height of stack.

- Largest out of place pancake.
  - Admissible?

- Manhattan distance.  
  - Admissible?
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
  - Largest out of place pancake.
    - Admissible? No. For number of flips.
    - Yes. For height of stack.
- Manhattan distance.
  - Admissible? Yes!
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
  - Admissible? Yes. For height of stack.

- Largest out of place pancake.
  - Admissible? No. For number of flips.
  - Admissible? Yes!
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.
  - Largest out of place pancake.
    - Admissible?
      - No. For number of flips.
      - Yes. For height of stack.

- Manhattan distance. Admissible?
  - Yes!

- Euclidean Distance.
Admissible Heuristics

A heuristic \( h \) is admissible (optimistic) if:

\[
0 \leq h(n) \leq h^*(n)
\]

where \( h^*(n) \) is the true cost to a nearest goal

Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

Optimize: number of flips.
Largest out of place pancake.
Admissible?
  No. For number of flips.
  Yes. For height of stack.

Manhattan distance.
Admissible? Yes!

Euclidean Distance.
Admissible?
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.

Note: Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Examples:

- Optimize: number of flips.  
  Admissible? No. For number of flips.
  Yes. For height of stack.

- Largest out of place pancake.  
  Admissible? Yes.

- Manhattan distance.  
  Admissible? Yes!

- Euclidean Distance.  
  Admissible? Yes.
Optimality of A* Tree Search
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n)$.

$\implies h = 0$ at a goal
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n)$

$\implies h = 0$ at a goal

Imagine $B$ is on the fringe.
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n)$

$\implies h = 0$ at a goal

Imagine B is on the fringe.

Some ancestor $n$ of A is on the fringe, too (maybe A!)
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$. 
Admissibility of $h(n) \leq h^*(n)$
$\implies h = 0$ at a goal
Imagine B is on the fringe.

Some ancestor $n$ of A is on the fringe, too (maybe A!)
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with \( g(B) > g(A) \).

Proof:

Definition of f-cost: \( g(n) + h(n) \).

Admissibility of \( h(n) \leq h^*(n) \)

\[ \implies h = 0 \text{ at a goal} \]

Imagine B is on the fringe.

Some ancestor \( n \) of A is on the fringe, too (maybe A!)

Claim: \( n \) will be expanded before B
Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n) \implies h = 0$ at a goal

Imagine $B$ is on the fringe.

Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)

Claim: $n$ will be expanded before $B$ $f(n)$ is less or equal to $f(A)$
Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$

$\implies h = 0$ at a goal

Imagine B is on the fringe.

Some ancestor $n$ of A is on the fringe, too (maybe A!)

Claim: $n$ will be expanded before B

$f(n)$ is less or equal to $f(A)$

since $f(n) = g(n) + h(n) < f(A)$,
$f(A)$ is less than $f(B)$
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n)$

$\implies h = 0$ at a goal

Imagine $B$ is on the fringe.

Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)

Claim: $n$ will be expanded before $B$

$f(n)$ is less or equal to $f(A)$

since $f(n) = g(n) + h(n) < f(A)$,

$f(A)$ is less than $f(B)$

since $g(A) + 0 < g(B) + h(B)$
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with \( g(B) > g(A) \).

Proof:

Definition of f-cost: \( g(n) + h(n) \).

Admissibility of \( h(n) \leq h^*(n) \)

\[ \implies h = 0 \text{ at a goal} \]

Imagine B is on the fringe.

Some ancestor \( n \) of A is on the fringe, too (maybe A!)

Claim: \( n \) will be expanded before B

- \( f(n) \) is less or equal to \( f(A) \)
  - since \( f(n) = g(n) + h(n) < f(A) \),
- \( f(A) \) is less than \( f(B) \)
  - since \( g(A) + 0 < g(B) + h(B) \)
Claim: Goal \( A \) is expanded before \( B \) with \( g(B) > g(A) \).

Proof:

Definition of f-cost: \( g(n) + h(n) \).

Admissibility of \( h(n) \leq h^*(n) \)

\[ \implies h = 0 \text{ at a goal} \]

Imagine \( B \) is on the fringe.

Some ancestor \( n \) of \( A \) is on the fringe, too (maybe \( A \)!

Claim: \( n \) will be expanded before \( B \)

\( f(n) \) is less or equal to \( f(A) \)

since \( f(n) = g(n) + h(n) < f(A) \),

\( f(A) \) is less than \( f(B) \)

since \( g(A) + 0 < g(B) + h(B) \)

All ancestors of \( A \) expand before \( B \).
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\implies h = 0$ at a goal

Imagine $B$ is on the fringe.

Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)

Claim: $n$ will be expanded before $B$

$f(n)$ is less or equal to $f(A)$

since $f(n) = g(n) + h(n) < f(A)$,

$f(A)$ is less than $f(B)$

since $g(A) + 0 < g(B) + h(B)$

All ancestors of $A$ expand before $B$.

$A$ expands before $B$. 
Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.

Admissibility of $h(n) \leq h^*(n)$

$\implies h = 0$ at a goal

Imagine B is on the fringe.

Some ancestor $n$ of A is on the fringe, too (maybe A!)

Claim: $n$ will be expanded before B

f(n) is less or equal to f(A)

since $f(n) = g(n) + h(n) < f(A)$,

f(A) is less than f(B)

since $g(A) + 0 < g(B) + h(B)$

All ancestors of A expand before B.

A expands before B. $\implies$ A* search is optimal.
Properties of $A^*$

UCS

A*
UCS vs A* Contours

Uniform-cost expands equally in all “directions”.

Goal

Goal
UCS vs A* Contours

Uniform-cost expands equally in all “directions”.

A* expands mainly toward the goal, but does hedge its bets to ensure optimality.

[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacmansmall maze (L3D5)]
Video of Demo Contours (Empty) – UCS
Video of Demo Contours (Empty) – Greedy
Video of Demo Contours (Empty) – A*
Video of Demo Contours: Pacman A*
Comparison

Uniform Cost

Greedy

A*
A* Applications
**A* Applications**

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]
Video of Demo Pacman (Tiny Maze) – UCS / A*
Video: Demo Water Shallow/Deep – Guess Algorithm

Total cost: 27
Number of nodes expanded: 187
Number of unique nodes expanded: 182
Pacman emerges victorious! Score: 979

{'numKills': 0, 'results': ['Miss'], 'numMoves': 27, 'score': 979}
Creating Heuristics
Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Manhattan Distance: 15

Straight Line Distance: 366
Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available.

Manhattan Distance: 15

Straight Line Distance: 366
Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to \textit{relaxed problems}, where new actions are available.

Inadmissible heuristics are often useful too.

\textbf{Manhattan Distance: 15}

\textbf{Straight Line Distance: 366}
Example: 8 Puzzle

Start State

Goal State

Actions
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?

What are the actions?

How many successors from the start state?

What should the costs be?
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?
How many states?

What are the actions?
How many successors from the start state?
What should the costs be?
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?
How many states?
What are the actions?

How many successors from the start state?
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?
How many states?
What are the actions?
How many successors from the start state?
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?
How many states?
What are the actions?
How many successors from the start state?
What should the costs be?
8 Puzzle 1

Heuristic: Number of tiles misplaced

Average nodes expanded when the optimal path has 4 steps 8 steps 12 steps.

UCS 112 6300 3

TILES 13 39 227
8 Puzzle I

Heuristic: Number of tiles misplaced

Why is it admissible? \( h(\text{start}) = \)
8 Puzzle I

Heuristic: Number of tiles misplaced
Why is it admissible? \( h(\text{start}) = \)
This is a relaxed-problem heuristic
8 Puzzle I

Heuristic: Number of tiles misplaced

Why is it admissible? $h(\text{start}) = \ldots$

This is a relaxed-problem heuristic

Average nodes expanded when the optimal path has
8 Puzzle I

Heuristic: Number of tiles misplaced

Why is it admissible? $h(\text{start}) =$

This is a relaxed-problem heuristic

Average nodes expanded when the optimal path has

<table>
<thead>
<tr>
<th></th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS</td>
<td>112</td>
<td>6300</td>
<td>$3.6 \times 10^6$</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

Start State

Goal
8 Puzzle II

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

Why is it admissible?

Start State  Goal
State
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

Why is it admissible?

\[ h(start) = 3 + 1 + 2 + \ldots = 18. \]
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

Why is it admissible?

\[ h(\text{start}) = 3 + 1 + 2 + \ldots = 18. \]

Average nodes expanded when the optimal path has:

<table>
<thead>
<tr>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</td>
<td>25</td>
</tr>
</tbody>
</table>
How about using the actual cost as a heuristic?
How about using the actual cost as a heuristic? Would it be admissible?
How about using the actual cost as a heuristic? Would it be admissible? Would we save on nodes expanded?
How about using the actual cost as a heuristic?
Would it be admissible?
Would we save on nodes expanded?
What’s wrong with it?
How about using the actual cost as a heuristic?
Would it be admissible?
Would we save on nodes expanded?
What’s wrong with it?

With A*: a trade-off between quality of estimate and work per node

- As heuristics get closer to the true cost, expand fewer nodes, but more work per node to compute the heuristic itself.
Semi-Lattice of Heuristics
Trivial Heuristics, Dominance

Dominance: \( h_a \geq h_c \) if

\[ \forall n: h_a(n) \geq h_c(n) \]

Heuristics form a semi-lattice:
- Max of admissible heuristics is admissible.
- Bottom of lattice is the zero heuristic.
- Top of lattice is the exact heuristic.
Trivial Heuristics, Dominance

Dominance: \( h_a \geq h_c \) if
\[
\forall n : h_a(n) \geq h_c(n)
\]

Heuristics form a semi-lattice:
Trivial Heuristics, Dominance

Dominance: $h_a \geq h_c$ if
\[ \forall n : h_a(n) \geq h_c(n) \]

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.
Trivial Heuristics, Dominance

\[ \text{Dominance: } h_a \geq h_c \text{ if } \forall n : h_a(n) \geq h_c(n) \]

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Trivial Heuristics, Dominance

Dominance: \( h_a \geq h_c \) if

\[
\forall n : h_a(n) \geq h_c(n)
\]

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.
Trivial Heuristics, Dominance

Dominance: \( h_a \geq h_c \) if

\[ \forall n: h_a(n) \geq h_c(n) \]

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.
(what does this give us?)
Trivial Heuristics, Dominance

\[
\text{max}(h_a, h_b) \quad \text{exact}
\]

Dominance: \( h_a \geq h_c \) if

\[ \forall n : h_a(n) \geq h_c(n) \]

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.
(what does this give us?)
Top of lattice is the exact heuristic
Graph Search
Tree Search: Extra Work!

Failure to detect repeated states can cause exponentially more work.
In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)
Graph Search

Idea: never expand a state twice
Graph Search

Idea: never expand a state twice

How to implement:

Tree search + set of expanded states ("closed set")
Expand the search tree node-by-node, but...
Before expanding a node, check if state was never been expanded before
If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness?

Why/why not?

How about optimality?
Graph Search

Idea: never expand a state twice

How to implement:
  Tree search + set of expanded states ("closed set")
Graph Search

Idea: never expand a state twice

How to implement:
  Tree search + set of expanded states (“closed set”)
  Expand the search tree node-by-node, but...
Graph Search

Idea: never expand a state twice

How to implement:
- Tree search + set of expanded states ("closed set")
- Expand the search tree node-by-node, but...
- Before expanding a node,
Graph Search

Idea: never expand a state twice

How to implement:
- Tree search + set of expanded states ("closed set")
- Expand the search tree node-by-node, but...
  - Before expanding a node,
    - check if state was never been expanded before
Graph Search

Idea: never expand a state twice

How to implement:
- Tree search + set of expanded states (“closed set”)
- Expand the search tree node-by-node, but...
  - Before expanding a node,
    - check if state was never been expanded before
  - If yes skip it, else add to closed set and expand.
Graph Search

Idea: never expand a state twice

How to implement:
Tree search + set of expanded states (“closed set”)  
Expand the search tree node-by-node, but...  
Before expanding a node,  
  check if state was never been expanded before  
  If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list
Graph Search

Idea: never expand a state twice

How to implement:
- Tree search + set of expanded states ("closed set")
- Expand the search tree node-by-node, but...
  - Before expanding a node,
    - check if state was never been expanded before
  - If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness?
Graph Search

Idea: never expand a state twice

How to implement:
- Tree search + set of expanded states ("closed set")
- Expand the search tree node-by-node, but...
  - Before expanding a node,
    - check if state was never been expanded before
  - If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?
Graph Search

Idea: never expand a state twice

How to implement:
  Tree search + set of expanded states (“closed set”)
  Expand the search tree node-by-node, but...
  Before expanding a node,
    check if state was never been expanded before
    If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?

How about optimality?
A* Graph Search Gone Wrong?

Is \( h(\cdot) \) admissible? Yes.

Will exploring w.r.t \( h(B) + g(n) \) be optimal?

Expand \( S \). A and B in fringe!

Expands B, since \( h(B) + g(B) = 2 < 5 = h(A) + g(A) \).

C in fringe with key, 3 + \( h(C) = 4 \).

G in fringe with key, 5.

Could have been there in 4.
A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible?
A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible? Yes.

---

**State space graph**

- **S** to **A**: $h=2$, $g=1$, total $=3$.
- **A** to **B**: $h=1$, $g=1$, total $=2$.
- **B** to **C**: $h=1$, $g=2$, total $=3$.
- **B** to **G**: $h=0$, $g=3$, total $=3$.
- **A** to **C**: $h=1$, $g=1$, total $=2$.
- **C** to **G**: $h=0$, $g=5$, total $=5$.

**Search tree**

- **S** $(0+2)$
  - **A** $(1+4)$
    - **C** $(2+1)$
      - **G** $(5+0)$
  - **B** $(1+1)$
    - **C** $(3+1)$
      - **G** $(6+0)$
A* Graph Search Gone Wrong?

Is \( h(\cdot) \) admissible? Yes.

Will exploring w.r.t \( h(B) + g(n) \) be optimal?
A* Graph Search Gone Wrong?

Is \( h(\cdot) \) admissible? Yes.

Will exploring w.r.t \( h(B) + g(n) \) be optimal?

Expand \( S \).
A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible? Yes.
Will exploring w.r.t $h(B) + g(n)$ be optimal?
Expand $S$.
  $A$ and $B$ in fringe!
A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible? Yes.

Will exploring w.r.t $h(B) + g(n)$ be optimal?

Expand $S$.
  - $A$ and $B$ in fringe!
  - Expands $B$, since $h(B) + g(B) = 2 < 5 = h(A) + g(A)$. 

A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible? Yes.

Will exploring w.r.t $h(B) + g(n)$ be optimal?

Expand $S$.
A and $B$ in fringe!
Expands $B$, since $h(B) + g(B) = 2 < 5 = h(A) + g(A)$.
$C$ in fringe with key, $3 + h(C) = 4$. 
A* Graph Search Gone Wrong?

Is \( h(\cdot) \) admissible? Yes.

Will exploring w.r.t \( h(B) + g(n) \) be optimal?

Expand \( S \).
  - \( A \) and \( B \) in fringe!
  - Expands \( B \), since \( h(B) + g(B) = 2 < 5 = h(A) + g(A) \).
  - \( C \) in fringe with key, \( 3 + h(C) = 4 \).
  - \( G \) in fringe with key, 5.
A* Graph Search Gone Wrong?

Is $h(\cdot)$ admissible? Yes.

Will exploring w.r.t $h(B) + g(n)$ be optimal?

Expand $S$.

$A$ and $B$ in fringe!

Expands $B$, since $h(B) + g(B) = 2 < 5 = h(A) + g(A)$.

$C$ in fringe with key, $3 + h(C) = 4$.

$G$ in fringe with key, $5$.

Could have been there in 4.
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

![Graph Diagram]

- From $A$ to $C$: $h = 1$ (cost = 1, heuristic = 4)
- From $C$ to $G$: $h = 2$ (cost = 3, heuristic = 3)

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x)$.
Proof: $f(y) = g(x) + \text{cost}(x, y) + h(y) \geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$. The "estimate" of plan cost keeps rising as you progress.
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq$ cost to goal.

![Graph showing nodes A, C, and G with heuristic costs h(A), h(C), and h(G) and edges with costs and labeled with h-values.]

Consistent $\Rightarrow$ admissible?

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + \text{cost}(x, y) + h(y) \geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq \text{cost}(x, y)$.
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq \text{cost}(x,y)$.

heuristic “arc” cost $\leq$ actual arc cost
**Consistency of Heuristics**

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq$ cost to goal.

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost $\leq$ actual arc cost

- $A \\ h = 1 \\ h = 4$
- $C \\ h = 2$
- $G \\ h = 3$
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

Admissibility: \( h(x) \leq \text{cost to goal} \).

Consistency: \( h(x) - h(y) \leq \text{cost}(x, y) \).
heuristic “arc” cost ≤ actual arc cost

Consistent \( \iff \) admissible?

![Graph with nodes A, C, and G and heuristic values]

\( h(A) = 1 \) 
\( h(C) = 2 \) 
\( h(G) = 1 \)
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal}. \)

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y). \)

heuristic “arc” cost ≤ actual arc cost

Consistent \( \implies \) admissible? Yes?

---

Graph:
- A
- C
- G

Edges:
- A to C: 1
- C to G: 3

Heuristic values:
- \( h(A) = 1 \)
- \( h(C) = 4 \)
- \( h(G) = 2 \)
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq$ cost to goal.

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent:
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal.} \)

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y). \)

heuristic “arc” cost ≤ actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** $h(x) \leq \text{cost to goal}.$

**Consistency:** $h(x) - h(y) \leq \text{cost}(x,y).$

heuristic “arc” cost ≤ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases
Admissible:
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq$ cost to goal.

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

\[ f(C) = h(C) + 1 = 3. \]
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** $h(x) \leq \text{cost to goal}$.

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost ≤ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3.0$

$f(A) = h(A) = 4.0$.
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal.} \)

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y). \)

heuristic “arc” cost ≤ actual arc cost

Consistent \( \iff \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

\[
\begin{align*}
    f(C) &= h(C) + 1 = 3. \\
    f(A) &= h(A) = 4. \\
\end{align*}
\]

Consistent:
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

Admissibility: \( h(x) \leq \text{cost to goal.} \)

Consistency: \( h(x) - h(y) \leq \text{cost}(x, y) \).

heuristic “arc” cost ≤ actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases
Admissible:
\[
\begin{align*}
  f(C) &= h(C) + 1 = 3. \\
  f(A) &= h(A) = 4.
\end{align*}
\]
Consistent: \( f(A) = 2 \)
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq$ cost to goal.

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y)$.

heuristic “arc” cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

- $f(C) = h(C) + 1 = 3$.
- $f(A) = h(A) = 4$.

Consistent: $f(A) = 2 < 3 = f(C)$. 

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + \text{cost}(x, y) + h(y) \geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x).$$
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** $h(x) \leq \text{cost to goal}.$

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y).$

heuristic “arc” cost ≤ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: $f$ value along a path never decreases

Admissible:

\[ f(C) = h(C) + 1 = 3. \]
\[ f(A) = h(A) = 4. \]

Consistent: $f(A) = 2 < 3 = f(C).$

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x).$
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal} \).

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y) \).

heuristic “arc” cost ≤ actual arc cost

Consistent \( \iff \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:
\[
\begin{align*}
    f(C) &= h(C) + 1 = 3. \\
    f(A) &= h(A) = 4. \\
    \text{Consistent: } f(A) &= 2 < 3 = f(C).
\end{align*}
\]

Claim: If \( y \) is expanded due to \( x \), \( f(y) \geq f(x) \).

Proof:
\[
f(y) = g(x) + \text{cost}(x, y) + h(y)
\]
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal}. \)

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y). \)

heuristic “arc” cost ≤ actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

\[
\begin{align*}
 f(C) &= h(C) + 1 = 3. \\
 f(A) &= h(A) = 4. \\
\end{align*}
\]

Consistent: \( f(A) = 2 < 3 = f(C). \)

Claim: If \( y \) is expanded due to \( x \), \( f(y) \geq f(x) \).

Proof:

\[
\begin{align*}
 f(y) &= g(x) + \text{cost}(x, y) + h(y) \\
 &\geq g(x) + h(x) - h(y) + h(y) \\
\end{align*}
\]
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal}. \)

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y). \)
heuristic “arc” cost ≤ actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:
\[
\begin{align*}
f(C) &= h(C) + 1 = 3. \\
f(A) &= h(A) = 4.
\end{align*}
\]

Consistent: \( f(A) = 2 < 3 = f(C) \).

Claim: If \( y \) is expanded due to \( x \), \( f(y) \geq f(x) \).

Proof:
\[
\begin{align*}
f(y) &= g(x) + \text{cost}(x, y) + h(y) \\
&\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x)
\end{align*}
\]
Consistency of Heuristics

Main idea: est. heuristic costs \( \leq \) actual costs

**Admissibility:** \( h(x) \leq \) cost to goal.

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y) \).

heuristic “arc” cost \( \leq \) actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

\[
\begin{align*}
f(C) &= h(C) + 1 = 3. \\
f(A) &= h(A) = 4. \\
\text{Consistent: } f(A) &= 2 < 3 = f(C).
\end{align*}
\]

Claim: If \( y \) is expanded due to \( x \), \( f(y) \geq f(x) \).

Proof:

\[
f(y) = g(x) + \text{cost}(x, y) + h(y)
\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)
\]
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

**Admissibility:** $h(x) \leq \text{cost to goal.}$

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y).$

heuristic “arc” cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: $f$ value along a path never decreases
Admissible:

$f(C) = h(C) + 1 = 3.$

$f(A) = h(A) = 4.$

Consistent: $f(A) = 2 < 3 = f(C).$

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x)$.

Proof:

$f(y) = g(x) + \text{cost}(x, y) + h(y)$

$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$

\[\square\]
Consistency of Heuristics

Main idea: est. heuristic costs ≤ actual costs

**Admissibility:** $h(x) \leq \text{cost to goal}.$

**Consistency:** $h(x) - h(y) \leq \text{cost}(x, y).$

heuristic “arc” cost ≤ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: $f$ value along a path never decreases

Admissible:

- $f(C) = h(C) + 1 = 3.$
- $f(A) = h(A) = 4.$

Consistent: $f(A) = 2 < 3 = f(C).$

Claim: If $y$ is expanded due to $x$, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + \text{cost}(x, y) + h(y) \geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$

The “estimate” of plan cost keeps rising as you progress.
Optimality of A* Graph Search
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state \( s \), the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state $s$, the optimal path is discovered.
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state $s$, the optimal path is discovered.

Result: A* graph search is optimal
Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state $s$, the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state $s$, the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$. 
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.
Proof: Consider first error.
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$. 
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$. 
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.
There is a vertex $v$ in the optimal path to $y$ in fringe.
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.
There is a vertex $v$ in the optimal path to $y$ in fringe.
$s$ in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$. 
Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.
There is a vertex $v$ in the optimal path to $y$ in fringe.
$s$ in fringe with key $f(s) = g(x) + \text{cost}(x, s) + h(s)$.
$v$ in fringe with key $f(v) = g(v) + h(v)$.
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.
There is a vertex $v$ in the optimal path to $y$ in fringe.
$s$ in fringe with key $f(s) = g(x) + \text{cost}(x, s) + h(s)$.
$v$ in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq \text{pathCost}(v, s)$ by induction.
Fact 2: The optimal path is discovered to every state \( s \).

Proof: Consider first error.
State \( s \) discovered from \( x \).
Optimal path is from \( y \neq x \).
There is a vertex \( v \) in the optimal path to \( y \) in fringe.
\( s \) in fringe with key \( f(s) = g(x) + \text{cost}(x, s) + h(s) \).
\( v \) in fringe with key \( f(v) = g(v) + h(v) \).
\( h(v) - h(s) \leq \text{pathCost}(v, s) \) by induction.
\( g(v) + \text{pathCost}(v, s) < g(x) + \text{cost}(x, s) \)
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.

There is a vertex $v$ in the optimal path to $y$ in fringe.

$s$ in fringe with key $f(s) = g(x) + \text{cost}(x,s) + h(s)$.

$v$ in fringe with key $f(v) = g(v) + h(v)$.

$h(v) - h(s) \leq \text{pathCost}(v,s)$ by induction.

$g(v) + \text{pathCost}(v,s) < g(x) + \text{cost}(x,s)$

$\implies f(v) < f(s)$. 
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error.
State $s$ discovered from $x$.
Optimal path is from $y \neq x$.
There is a vertex $v$ in the optimal path to $y$ in fringe.
$s$ in fringe with key $f(s) = g(x) + \text{cost}(x, s) + h(s)$.
$v$ in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq \text{pathCost}(v, s)$ by induction.
$g(v) + \text{pathCost}(v, s) < g(x) + \text{cost}(x, s)$
$\implies f(v) < f(s)$.
But then $v$ would have been expanded before $s$!
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state \( s \).

Proof: Consider first error. State \( s \) discovered from \( x \).

Optimal path is from \( y \neq x \).

There is a vertex \( v \) in the optimal path to \( y \) in fringe. 
\( s \) in fringe with key \( f(s) = g(x) + \text{cost}(x, s) + h(s) \).
\( v \) in fringe with key \( f(v) = g(v) + h(v) \).

\( h(v) - h(s) \leq \text{pathCost}(v, s) \) by induction.

\[ g(v) + \text{pathCost}(v, s) < g(x) + \text{cost}(x, s) \]
\[ \implies f(v) < f(s). \]

But then \( v \) would have been expanded before \( s \)!
Optimality

Tree search:
A* is optimal if heuristic is admissible.
Optimality

Tree search:
A* is optimal if heuristic is admissible.
UCS is a special case (h = 0)
Optimality

Tree search:
   A* is optimal if heuristic is admissible.
   UCS is a special case (h = 0)

Graph search:
Optimality

Tree search:
A* is optimal if heuristic is admissible.
UCS is a special case (h = 0)

Graph search:
A* optimal if heuristic is consistent
Optimality

Tree search:
  A* is optimal if heuristic is admissible.
  UCS is a special case (h = 0)

Graph search:
  A* optimal if heuristic is consistent
  UCS optimal (h = 0 is consistent)
Optimality

Tree search:
  A* is optimal if heuristic is admissible.
  UCS is a special case (h = 0)

Graph search:
  A* optimal if heuristic is consistent
  UCS optimal (h = 0 is consistent)

Consistency implies admissibility
Optimality

Tree search:
- A* is optimal if heuristic is admissible.
- UCS is a special case \((h = 0)\)

Graph search:
- A* optimal if heuristic is consistent
- UCS optimal \((h = 0\) is consistent)

Consistency implies admissibility

In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems
A*: Summary

A* uses both backward costs and (estimates of) forward costs.

A* is optimal with admissible / consistent heuristics.

Heuristic design is key: often use relaxed problems.
A*: Summary

A* uses both backward costs and (estimates of) forward costs
A*: Summary

A* uses both backward costs and (estimates of) forward costs
A* is optimal with admissible / consistent heuristics
A*: Summary

A* uses both backward costs and (estimates of) forward costs
A* is optimal with admissible / consistent heuristics
Heuristic design is key: often use relaxed problems
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
end
function \textsc{Graph-Search}(\textit{problem, fringe}) \textbf{return} a solution, or failure
   \begin{itemize}
      \item \textit{closed} $\leftarrow$ an empty set
      \item \textit{fringe} $\leftarrow$ \textsc{Insert}(\textsc{Make-Node}(\text{INITIAL-STATE}[\textit{problem}]), \textit{fringe})
      \item \textbf{loop} do
         \begin{itemize}
            \item if \textit{fringe} is empty then \textbf{return} failure
            \item \textit{node} $\leftarrow$ \textsc{Remove-Front}(\textit{fringe})
            \item if \textsc{Goal-Test}(\textit{problem, state}[\textit{node}]) then \textbf{return} \textit{node}
            \item if \text{STATE}[\textit{node}] is not in \textit{closed} then
               \begin{itemize}
                  \item add \text{STATE}[\textit{node}] to \textit{closed}
                  \item for \textit{child-node} in \textsc{Expand}(\text{STATE}[\textit{node}], \textit{problem}) do
                     \begin{itemize}
                        \item \textit{fringe} $\leftarrow$ \textsc{Insert}(\textit{child-node}, \textit{fringe})
                     \end{itemize}
               \end{itemize}
            \end{itemize}
         \item end
      \item end
   \end{itemize}
Yaay!