Heuristics in the A* Search

Outline

I. Properties of Heuristics

II. Comparison of Heuristics

* Figures are from the textbook site (or drawn by the instructor) unless the source is specifically cited.

I. Properties of Heuristics

- A* search is *complete* (when the state space either has a solution or is finite).
- Whether it is optimal depends on the heuristic.

I. Properties of Heuristics

- A* search is *complete* (when the state space either has a solution or is finite).
- Whether it is optimal depends on the heuristic.

A heuristic is *admissible* if it *never overestimates* the cost to reach a goal.

I. Properties of Heuristics

- A* search is *complete* (when the state space either has a solution or is finite).
- Whether it is optimal depends on the heuristic.

A heuristic is *admissible* if it *never overestimates* the cost to reach a goal.

h_{SLD}: straight-line distance

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

I. Properties of Heuristics

- A* search is *complete* (when the state space either has a solution or is finite).
- Whether it is optimal depends on the heuristic.

A heuristic is *admissible* if it *never overestimates* the cost to reach a goal.

h_{SLD}: straight-line distance

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

h_{SLD} is admissible because the actual distance to Bucharest cannot be less than the straight-line distance.

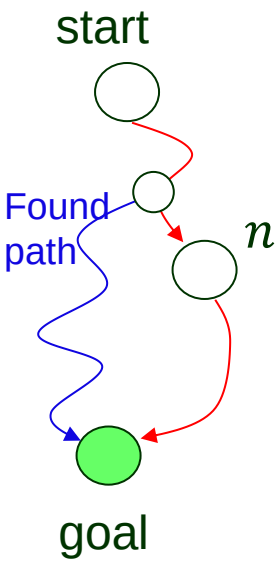
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.



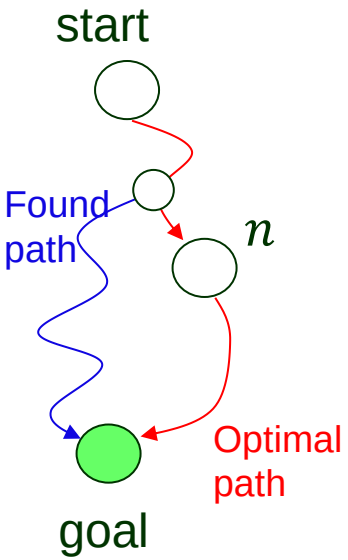
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



Optimality of A*

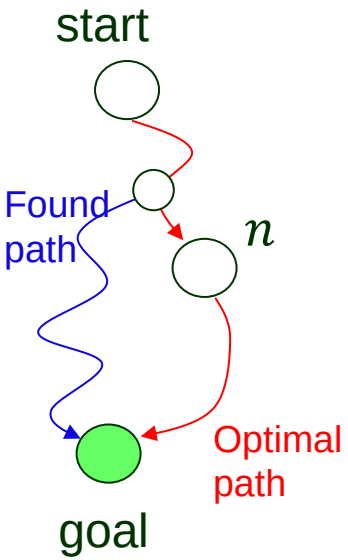
Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.

$g^*(n)$: optimal cost from start to n



Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

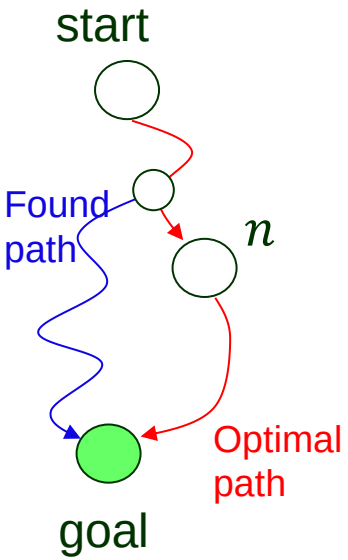
Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.

$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal



Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.

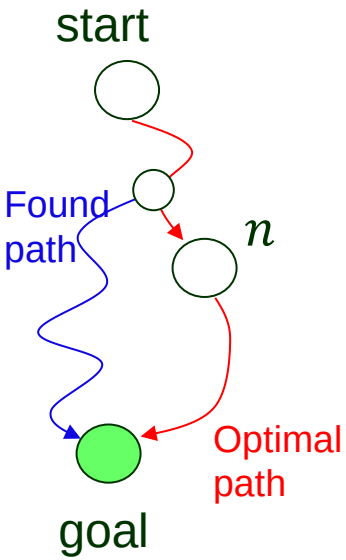
$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)



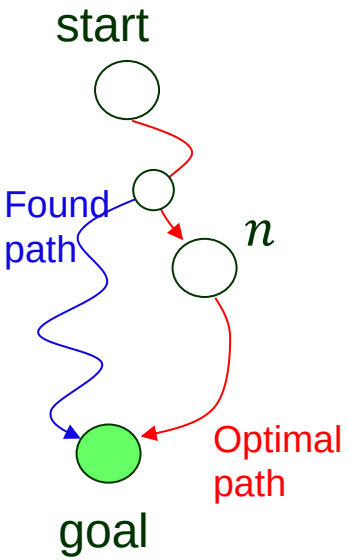
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

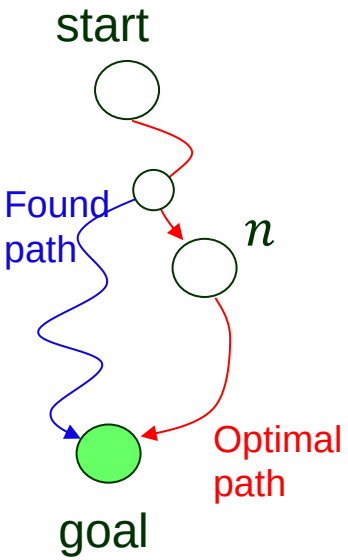
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ (n on the optimal path)

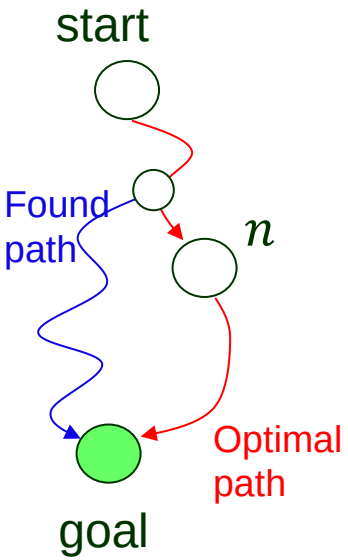
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ (n on the optimal path)

$\leq g^*(n) + h^*(n)$ ($h(n) \leq h^*(n)$ due to admissibility)

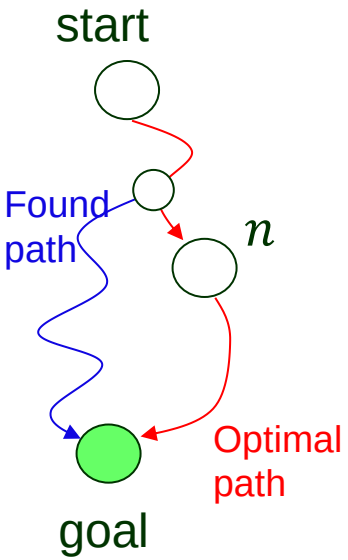
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ (n on the optimal path)

$\leq g^*(n) + h^*(n)$ ($h(n) \leq h^*(n)$ due to admissibility)

$= C^*$

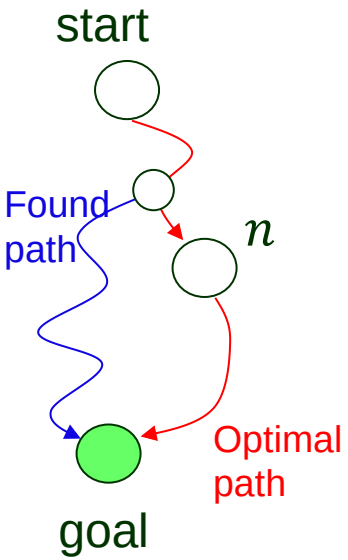
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ (n on the optimal path)

$\leq g^*(n) + h^*(n)$ ($h(n) \leq h^*(n)$ due to admissibility)

$= C^*$

That is, $f(n) \leq C^*$, contradicting with $f(n) > C^*$.

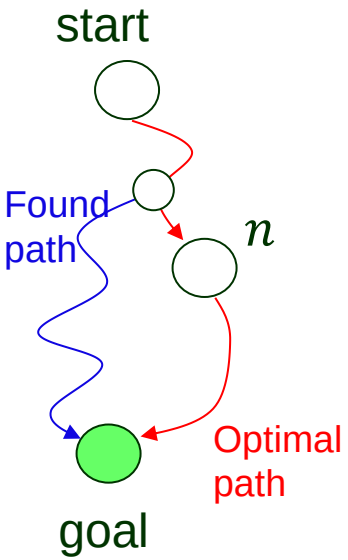
Optimality of A*

Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost C greater than the optimal cost C^*.

⇩

Let n be the first node on the optimal path that is *unexpanded*.



$g^*(n)$: optimal cost from start to n

$h^*(n)$: optimal cost from n to a goal

⇩

$f(n) > C^*$

(otherwise $f(n) \leq C^* < C$ so n would have been expanded before the goal node)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ (n on the optimal path)

$\leq g^*(n) + h^*(n)$ ($h(n) \leq h^*(n)$ due to admissibility)

$= C^*$

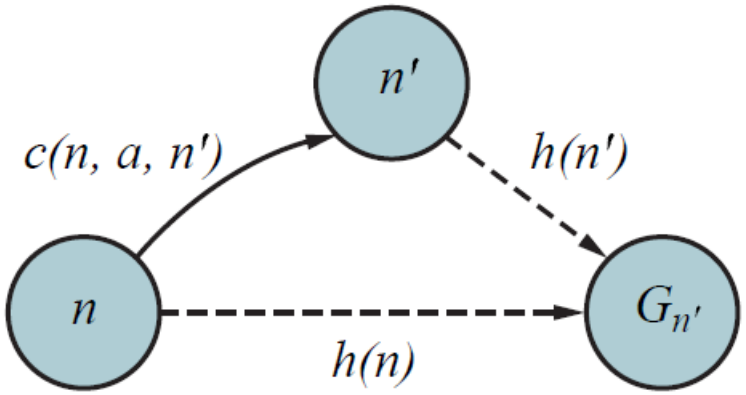
That is, $f(n) \leq C^*$, contradicting with $f(n) > C^*$.

Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

(*triangle inequality*)

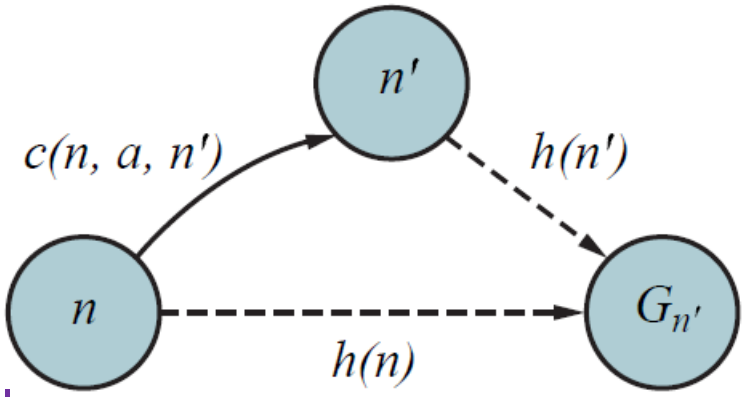


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

(*triangle inequality*)



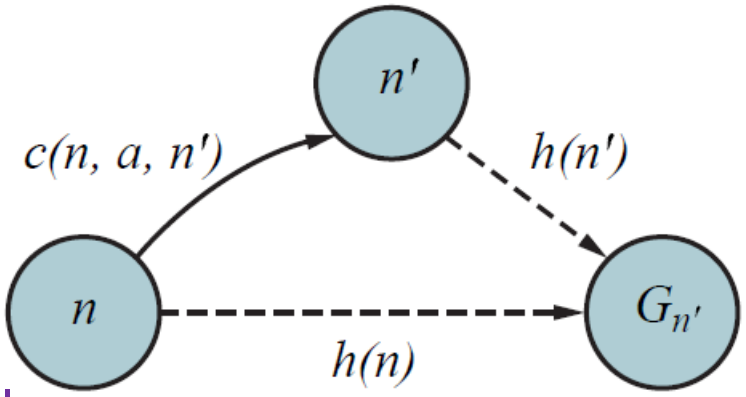
- Every consistent heuristic is admissible.

Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

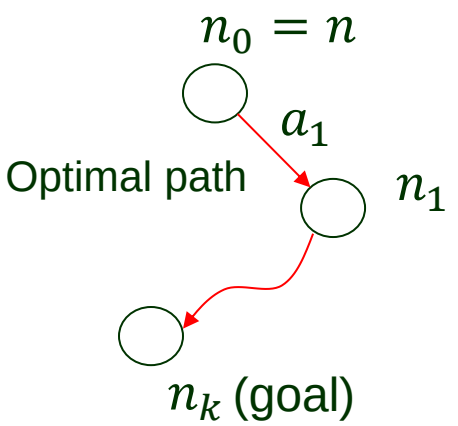
$$h(n) \leq c(n, a, n') + h(n')$$

(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

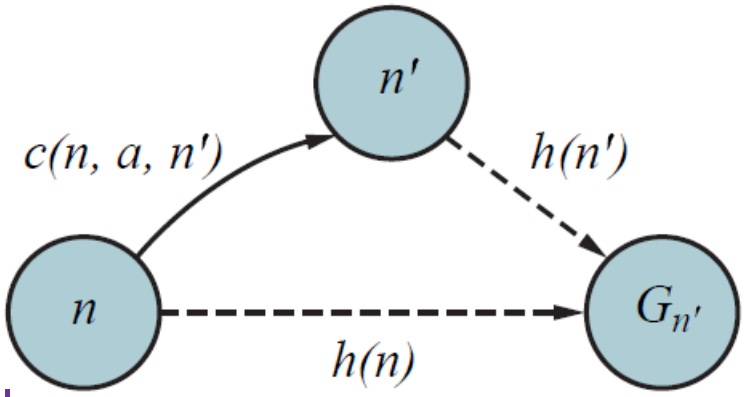


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

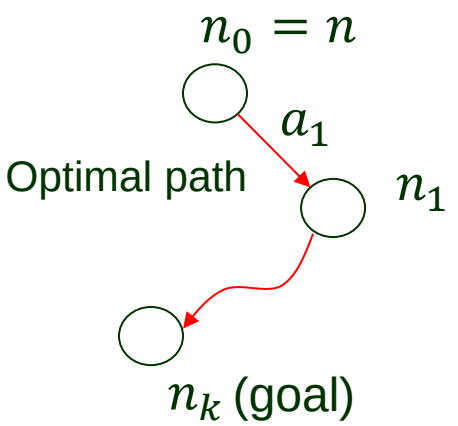
(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

  $$h(n) = h(n_0) \leq c(n_0, a_1, n_1) + h(n_1)$$

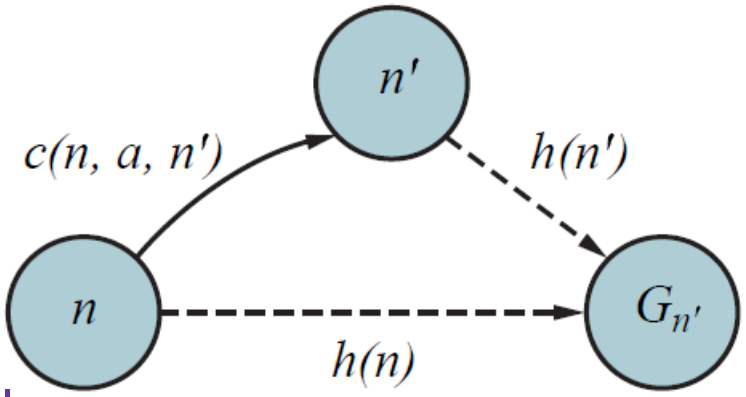


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

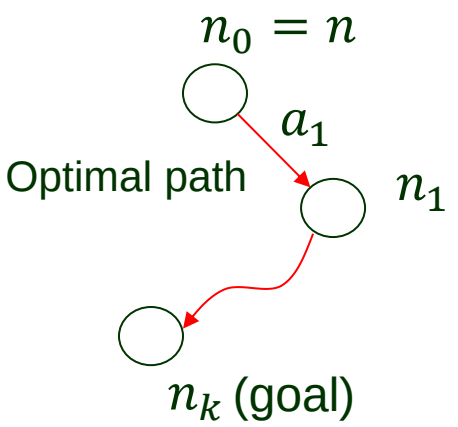
(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

$$h(n) = h(n_0) \leq c(n_0, a_1, n_1) + h(n_1)$$
$$\leq c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2)$$

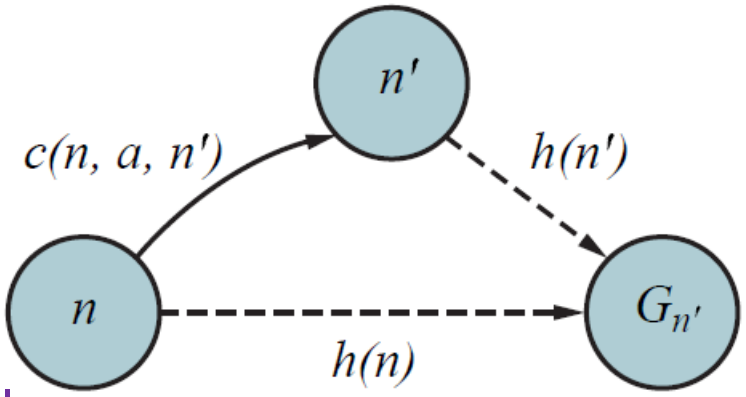


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

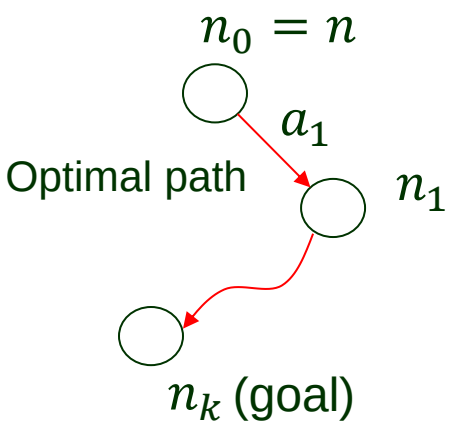
(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

$$\begin{aligned} h(n) = h(n_0) &\leq c(n_0, a_1, n_1) + h(n_1) \\ &\leq c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2) \\ &\vdots \\ &\leq \sum_{i=1}^{k} c(n_{i-1}, a_i, n_i) + h(n_k) \end{aligned}$$

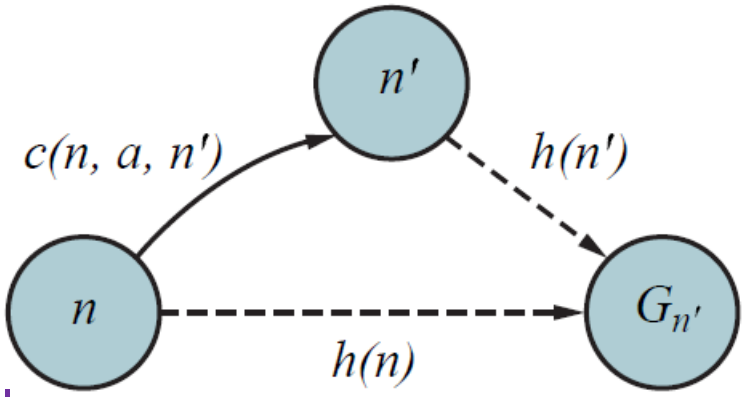


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

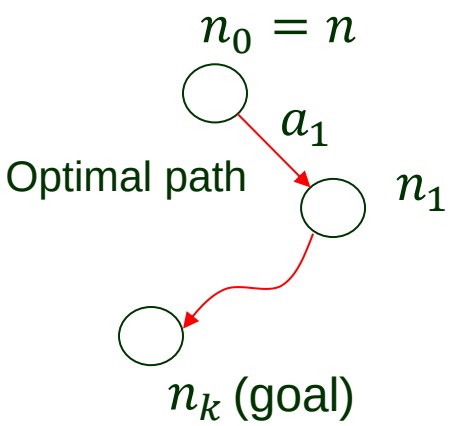
(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

$$\begin{aligned} h(n) = h(n_0) &\leq c(n_0, a_1, n_1) + h(n_1) \\ &\leq c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2) \\ &\vdots \\ &\leq \sum_{i=1}^{k} c(n_{i-1}, a_i, n_i) + \underbrace{h(n_k)}_{=0} \end{aligned}$$

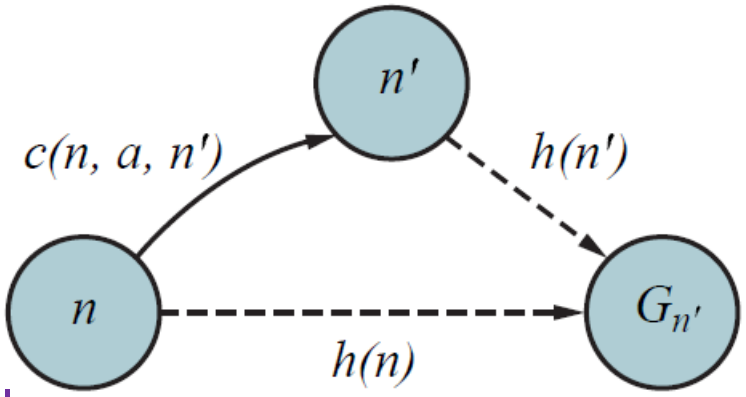


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

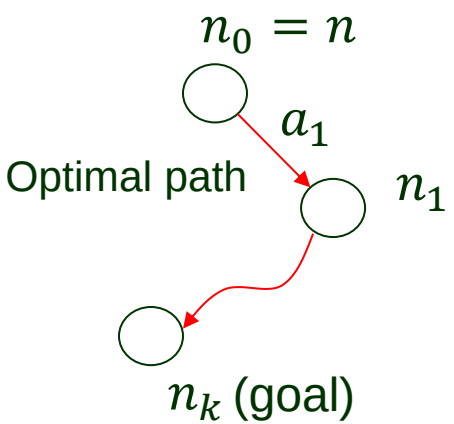
(*triangle inequality*)



- Every consistent heuristic is admissible.

Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

$$\begin{aligned} h(n) = h(n_0) &\leq c(n_0, a_1, n_1) + h(n_1) \\ &\leq c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2) \\ &\vdots \\ &\leq \sum_{i=1}^{k} c(n_{i-1}, a_i, n_i) + \underbrace{h(n_k)}_{=0} = h^*(n) \end{aligned}$$

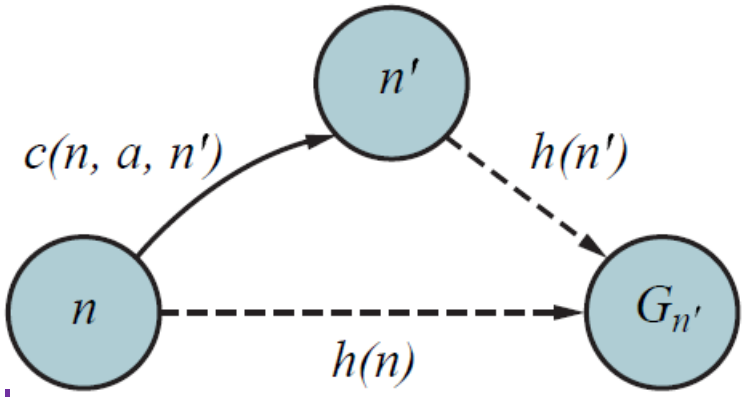


Consistent Heuristic

A heuristic h is *consistent* if for every two nodes n and n' such that n' is generated from n by some action a, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

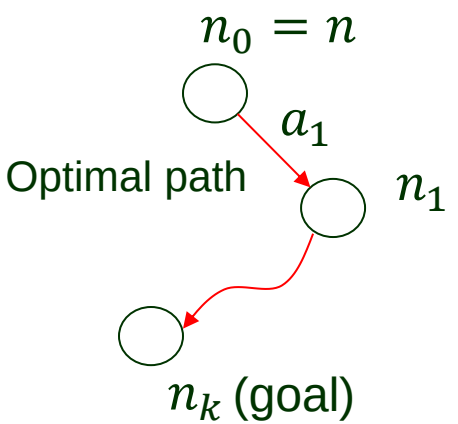
(*triangle inequality*)



- Every consistent heuristic is admissible.

  Consider the optimal path $n_0 = n, n_1, \ldots, n_k$ (goal).

$$\begin{aligned} h(n) = h(n_0) &\leq c(n_0, a_1, n_1) + h(n_1) \\ &\leq c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2) \\ &\vdots \\ &\leq \sum_{i=1}^{k} c(n_{i-1}, a_i, n_i) + \underbrace{h(n_k)}_{=0} = h^*(n) \end{aligned}$$

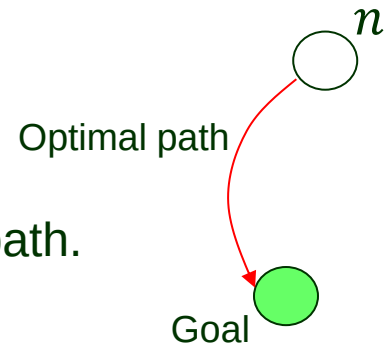


- A* with a consistent heuristic is cost-optimal.

What If the Heuristic Is Inadmissible?

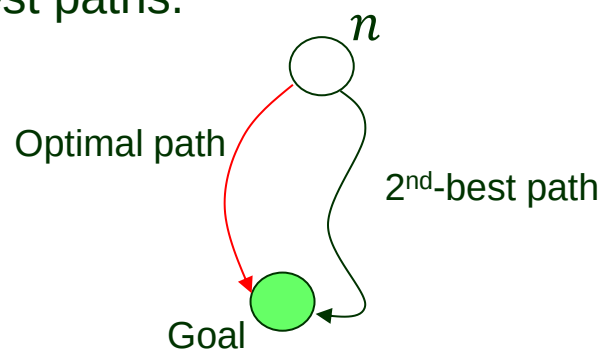
In such a situation, A* may or may not be cost-optimal.

Two of the cases where A* finds an optimal path:



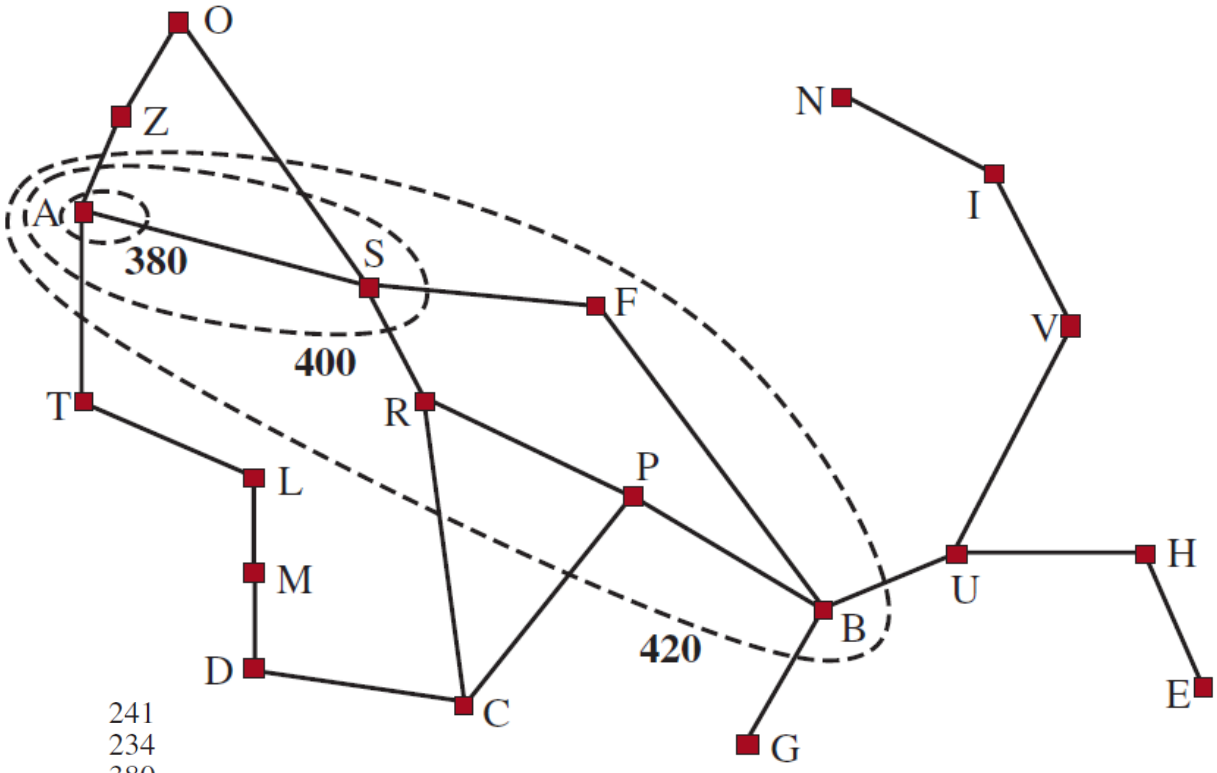
- h is admissible for all the nodes on one optimal path.

- $h(n)$ does not overestimate the cost on each node n by more than the difference between the costs of the optimal and the second-best paths.



Search Contours

A *contour* labeled by a cost c encloses all the nodes n with $f(n) = g(n) + h(n) \leq c$.

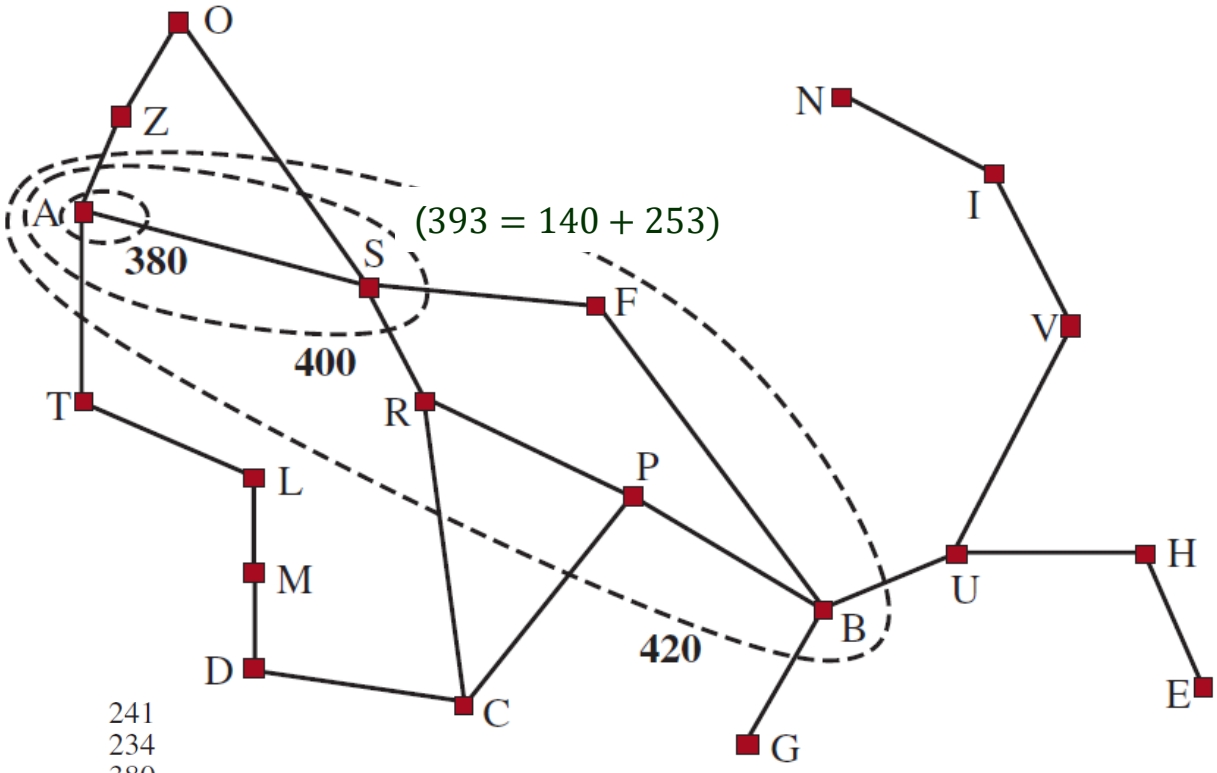


h_{SLD}: straight-line distance

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Search Contours

A *contour* labeled by a cost c encloses all the nodes n with $f(n) = g(n) + h(n) \leq c$.



h_{SLD}: straight-line distance

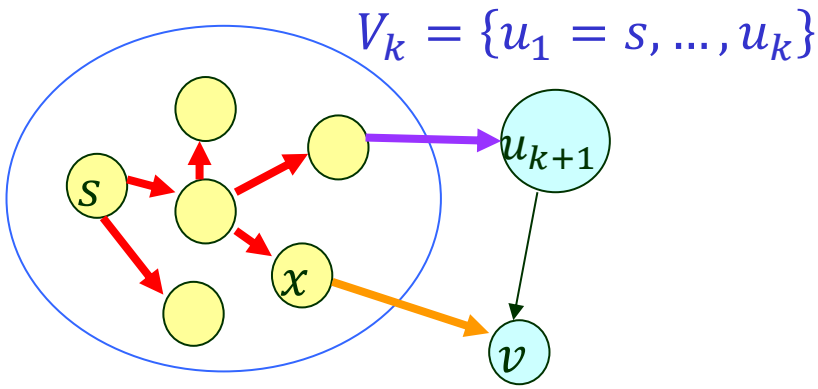
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Some Search Contours

- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.

Some Search Contours

- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.

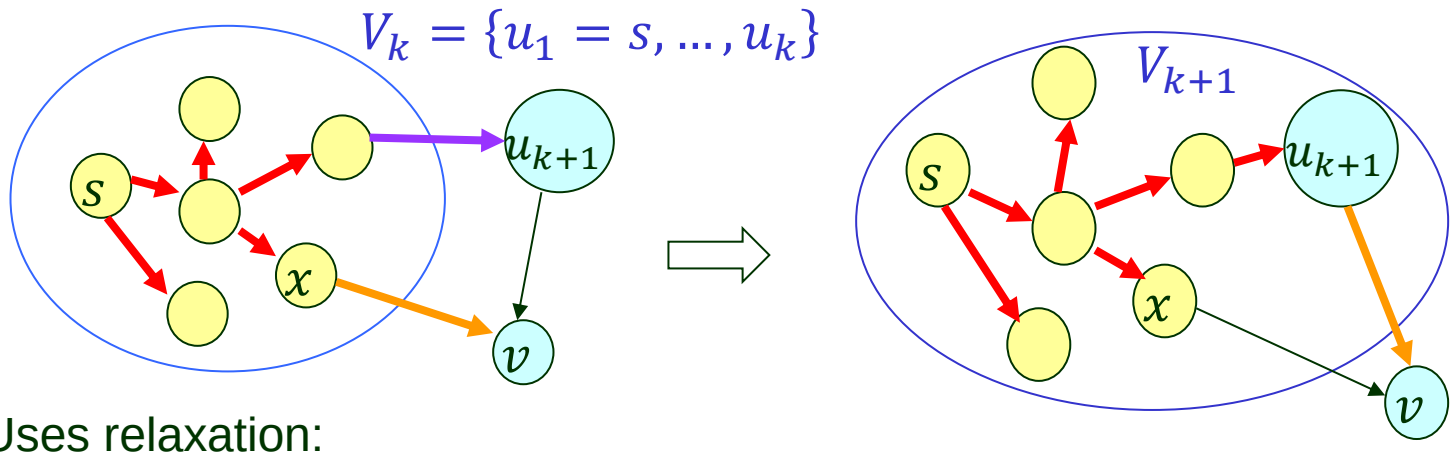


Uses relaxation:

$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

Some Search Contours

- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.

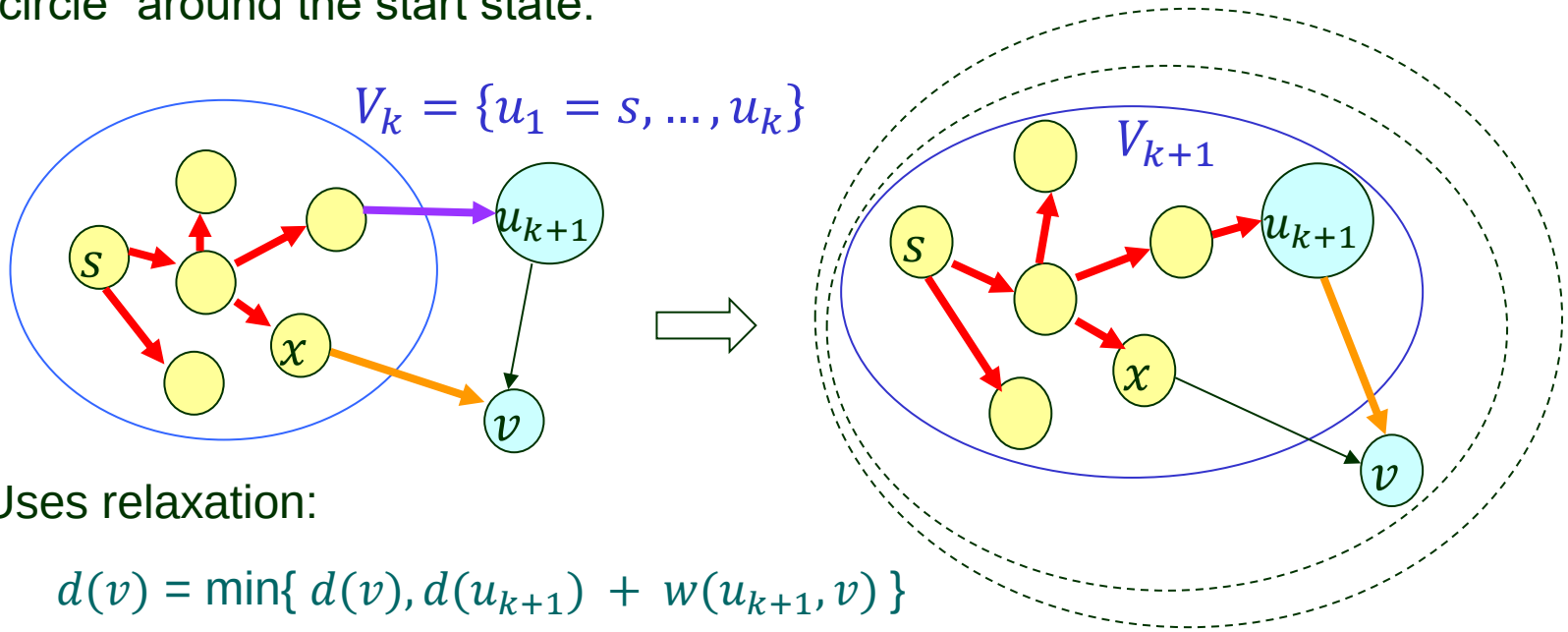


Uses relaxation:

$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

Some Search Contours

- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.

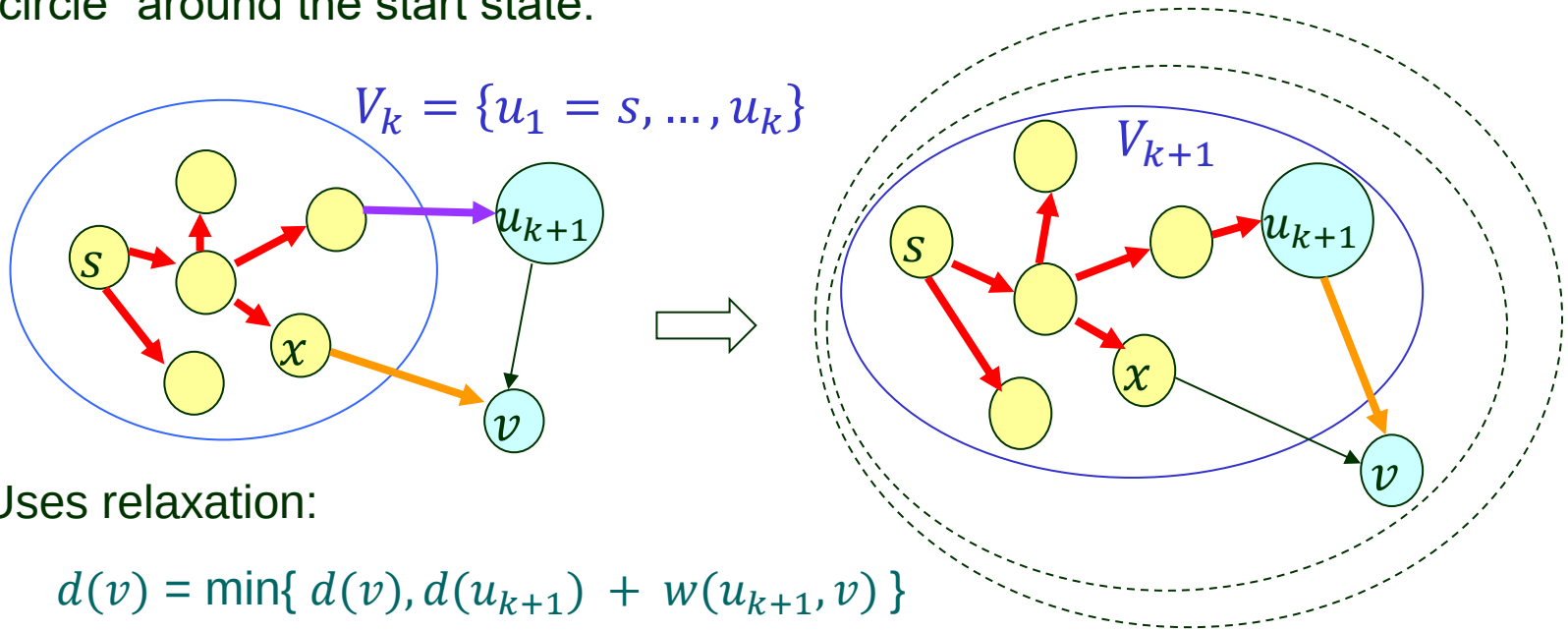


Uses relaxation:

$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

Some Search Contours

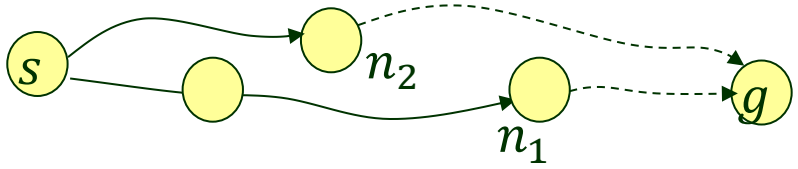
- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.



Uses relaxation:

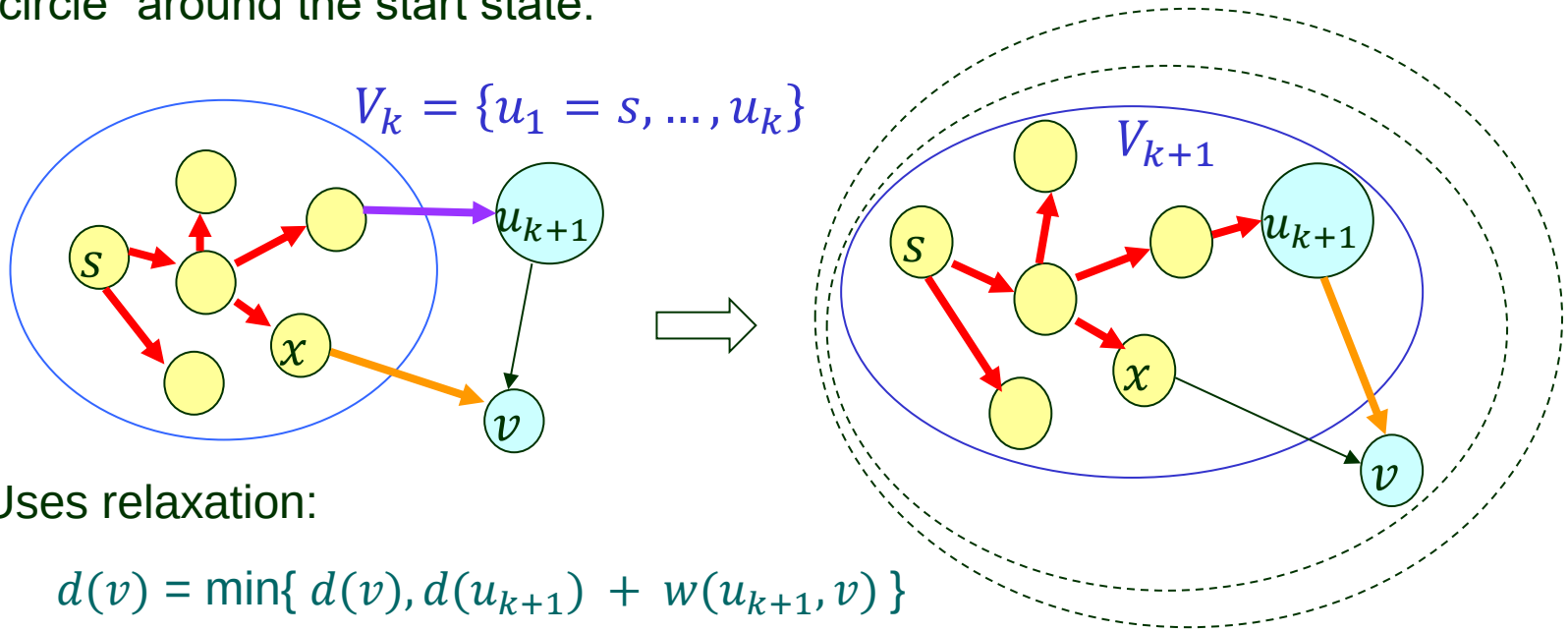
$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

- With a good h, the $g + h$ bands will stretch toward a goal state.



Some Search Contours

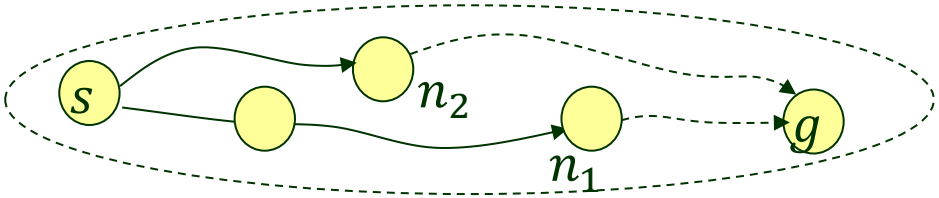
- Dijkstra's algorithm (i.e., uniform search) would have contours of g-cost to "circle" around the start state.



Uses relaxation:

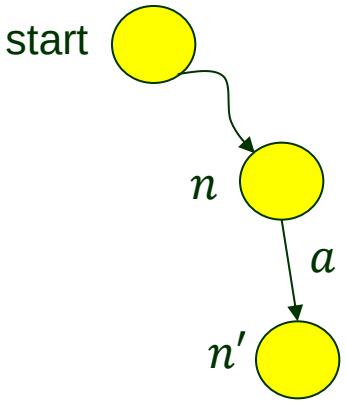
$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

- With a good h, the $g + h$ bands will stretch toward a goal state.



Monotonicity?

- The g cost increases along a path because action costs are positive.

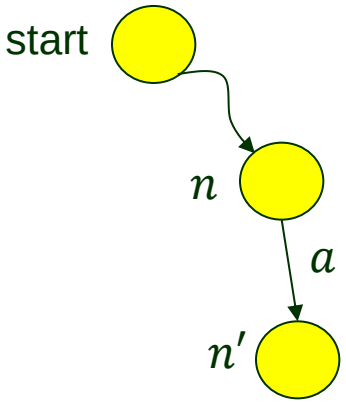


Cost at n: $g(n) + h(n)$

Cost at its successor n': $\underbrace{g(n) + c(n, a, n')}_{= g(n')} + h(n')$

Monotonicity?

- The g cost increases along a path because action costs are positive.



Cost at n: $g(n) + h(n)$

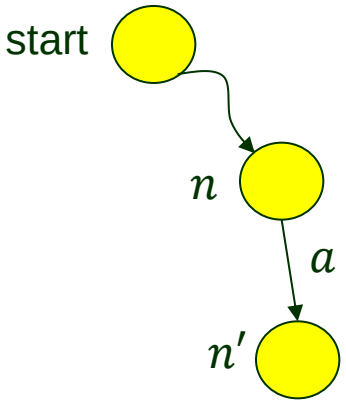
Cost at its successor n': $\underbrace{g(n) + c(n, a, n')}_{= g(n')} + h(n')$

The path's cost increases *monotonically* iff

$$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$$

Monotonicity?

- The g cost increases along a path because action costs are positive.



Cost at n: $g(n) + h(n)$

Cost at its successor n': $\underbrace{g(n) + c(n, a, n')}_{= g(n')} + h(n')$

The path's cost increases *monotonically* iff

$$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$$

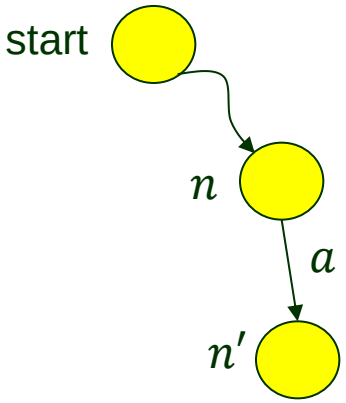
$$\Updownarrow$$

$$h(n) \leq c(n, a, n') + h(n')$$

(consistent heuristic)

Monotonicity?

- The g cost increases along a path because action costs are positive.



Cost at n: $g(n) + h(n)$

Cost at its successor n': $\underbrace{g(n) + c(n, a, n')}_{= g(n')} + h(n')$

The path's cost increases *monotonically* iff

$$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$$

$$\Updownarrow$$

$$h(n) \leq c(n, a, n') + h(n')$$

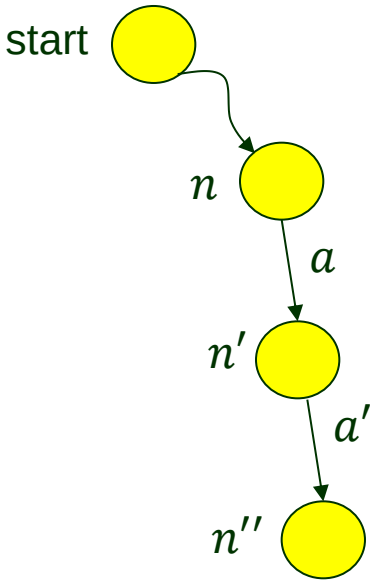
(consistent heuristic)

Monotonicity is equivalent to consistency for a heuristic.

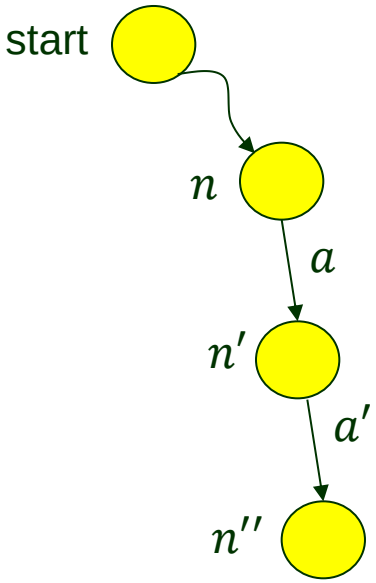
When Consecutive Nodes Are Scored the Same

$$g(n) + h(n) = \overbrace{g(n) + c(n, a, n')}^{= g(n')} + h(n')$$

$$= \underbrace{g(n) + c(n, a, n') + c(n', a', n'')}_{= g(n'')} + h(n'')$$



When Consecutive Nodes Are Scored the Same



$$g(n) + h(n) = \overbrace{g(n) + c(n, a, n')}^{= g(n')} + h(n')$$

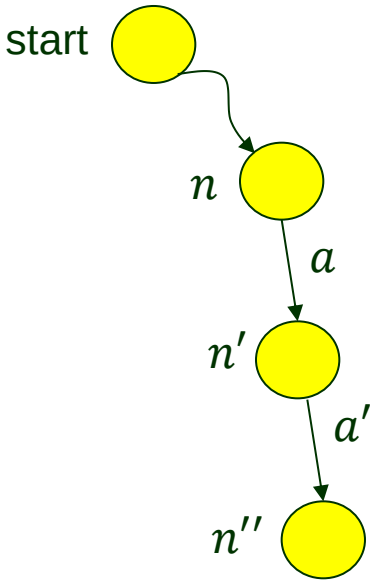
$$= \underbrace{g(n) + c(n, a, n') + c(n', a', n'')}_{= g(n'')} + h(n'')$$

$$\Updownarrow$$

$$h(n) - h(n') = g(n') - g(n)$$

$$h(n') - h(n'') = g(n'') - g(n')$$

When Consecutive Nodes Are Scored the Same



$$g(n) + h(n) = \overbrace{g(n) + c(n, a, n')}^{= g(n')} + h(n')$$

$$= \underbrace{g(n) + c(n, a, n') + c(n', a', n'')}_{= g(n'')} + h(n'')$$

$$\Updownarrow$$

$$h(n) - h(n') = g(n') - g(n)$$
$$h(n') - h(n'') = g(n'') - g(n')$$

h decreases as much as g increases after an action.

Efficiency of A*

h: admissible

C^*: cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs $< C^*$.
- A* will not expand any node n with $f(n) > C^*$.
- A* might expand a node n with cost $f(n) = C^*$ before selecting a goal node.

Efficiency of A*

h: admissible

C^*: cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs $< C^*$.
- A* will not expand any node n with $f(n) > C^*$.
- A* might expand a node n with cost $f(n) = C^*$ before selecting a goal node.

A* prunes away nodes unnecessary for finding an optimal solution.

Efficiency of A*

h: admissible

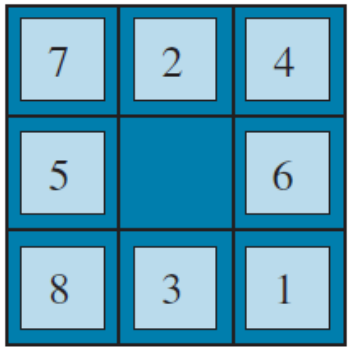
C^*: cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs $< C^*$.
- A* will not expand any node n with $f(n) > C^*$.
- A* might expand a node n with cost $f(n) = C^*$ before selecting a goal node.

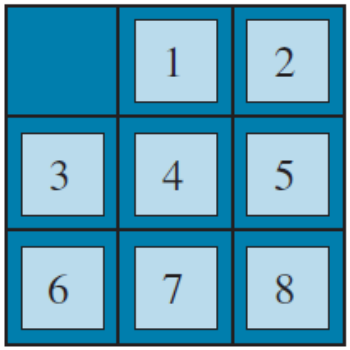
A* prunes away nodes unnecessary for finding an optimal solution.

- A* may take exponential time with a poor heuristic function.

II. 8-Puzzle: Large Search Space

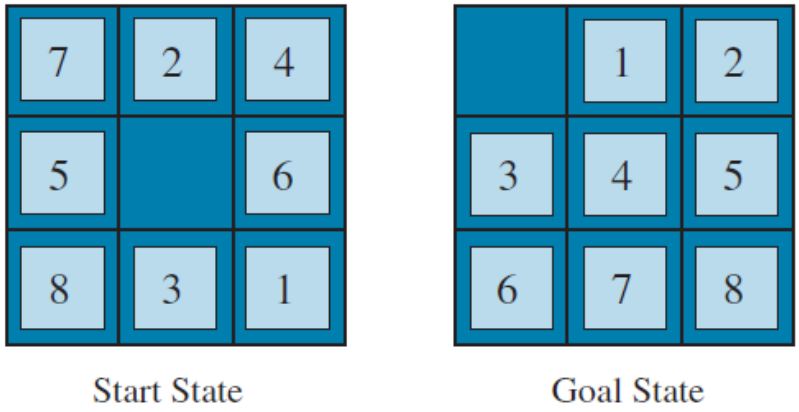


Start State



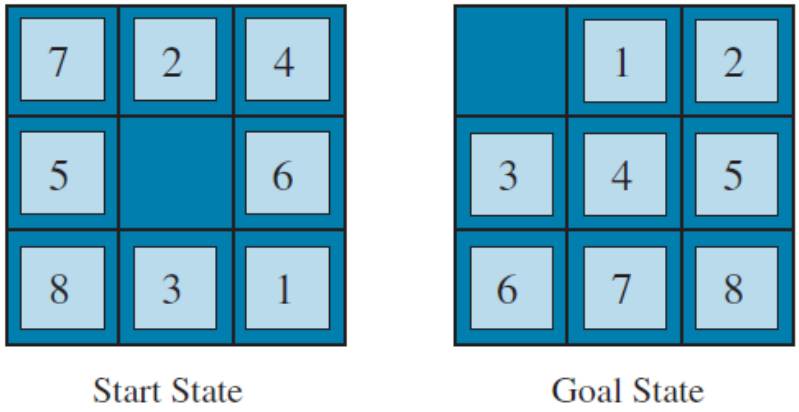
Goal State

II. 8-Puzzle: Large Search Space



$\frac{9!}{2} = 181{,}400$ reachable states from start.

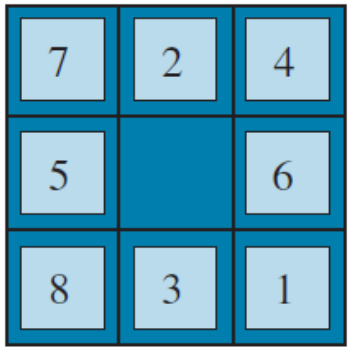
II. 8-Puzzle: Large Search Space



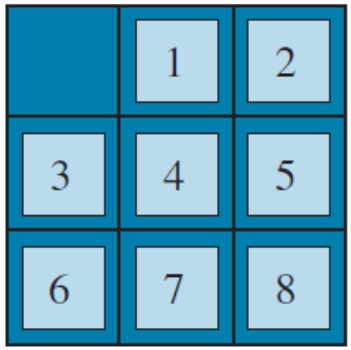
$\frac{9!}{2} = 181{,}400$ reachable states from start.

$\frac{16!}{2} > 10^{13}$ reachable states for the 15-puzzle!

Two Heuristics



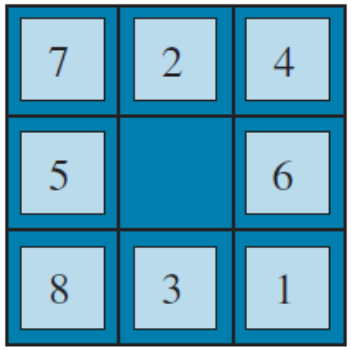
Start State



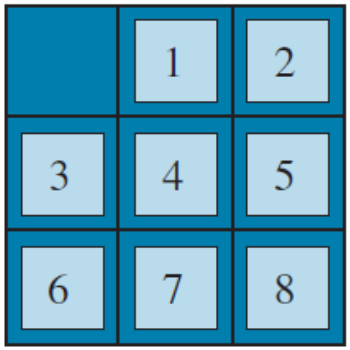
Goal State

Heuristics are needed for searching the vast state space.

Two Heuristics



Start State

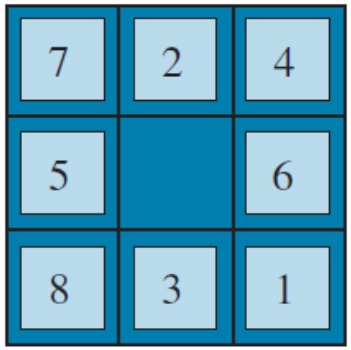


Goal State

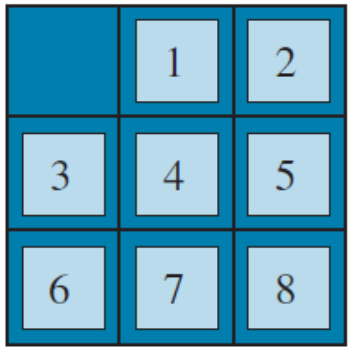
Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

Two Heuristics



Start State



Goal State

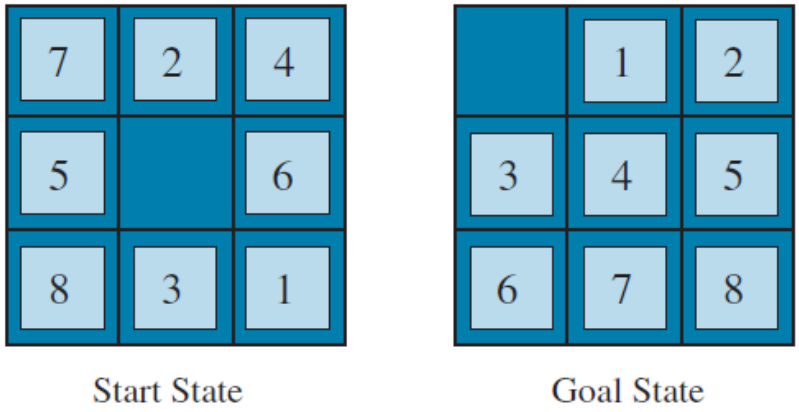
Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

Misplaced tiles:

1, 2, 3, 4, 5, 6, 7, 8

Two Heuristics

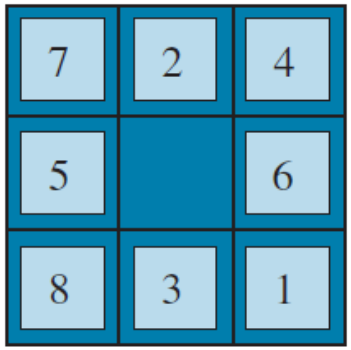


Heuristics are needed for searching the vast state space.

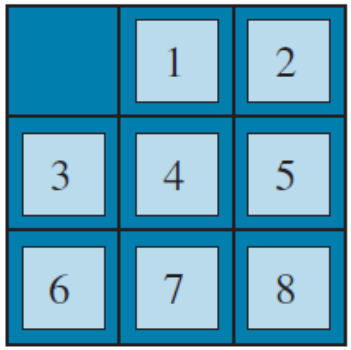
- h_1 = # tiles misplaced

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 $\Rightarrow$ $h_1 = 8$

Two Heuristics



Start State



Goal State

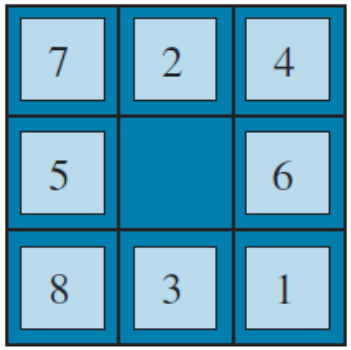
Heuristics are needed for searching the vast state space.

♦ h_1 = # tiles misplaced

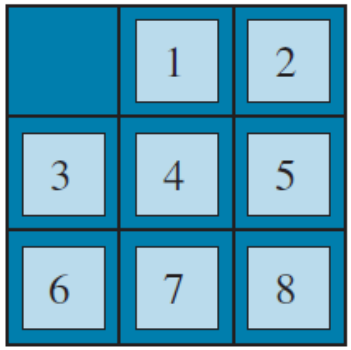
Admissible: any tile out of place will require ≥ 1 move to fix.

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

Two Heuristics



Start State



Goal State

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

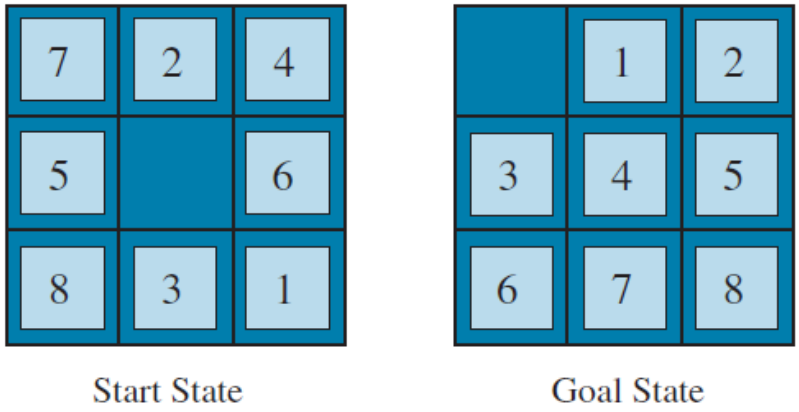
Heuristics are needed for searching the vast state space.

♦ h_1 = # tiles misplaced

Admissible: any tile out of place will require ≥ 1 move to fix.

♦ h_2 = sum of Manhattan distances of the tiles from their goal positions

Two Heuristics



Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

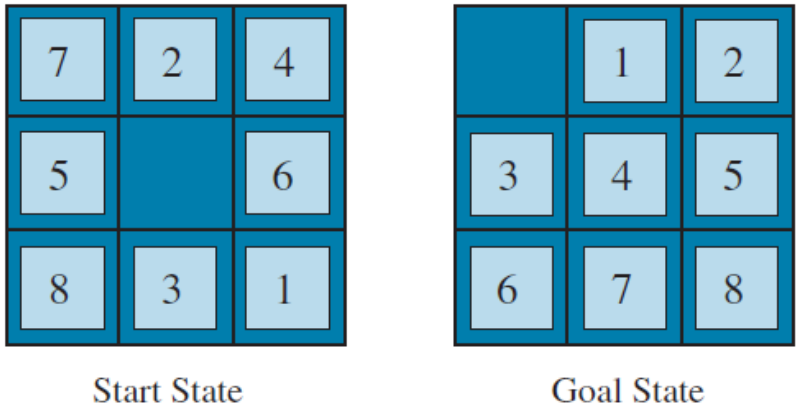
  Admissible: any tile out of place will require ≥ 1 move to fix.

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

Tile	1	2	3	4	5	6	7	8
Manhattan distance	3	1	2	2	2	3	3	2

- h_2 = sum of Manhattan distances of the tiles from their goal positions

Two Heuristics



Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

  Admissible: any tile out of place will require ≥ 1 move to fix.

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

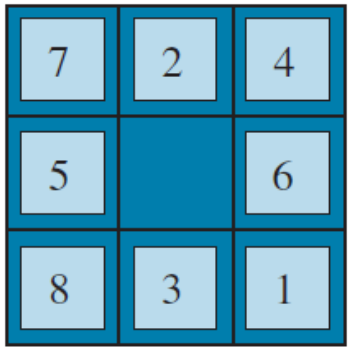
Tile	1	2	3	4	5	6	7	8
Manhattan distance	3	1	2	2	2	3	3	2

⇩

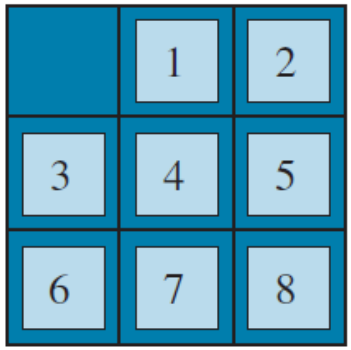
$h_2 = 18$

- h_2 = sum of Manhattan distances of the tiles from their goal positions

Two Heuristics



Start State



Goal State

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

Tile	1	2	3	4	5	6	7	8
Manhattan distance	3	1	2	2	2	3	3	2

⇩

$h_2 = 18$

Heuristics are needed for searching the vast state space.

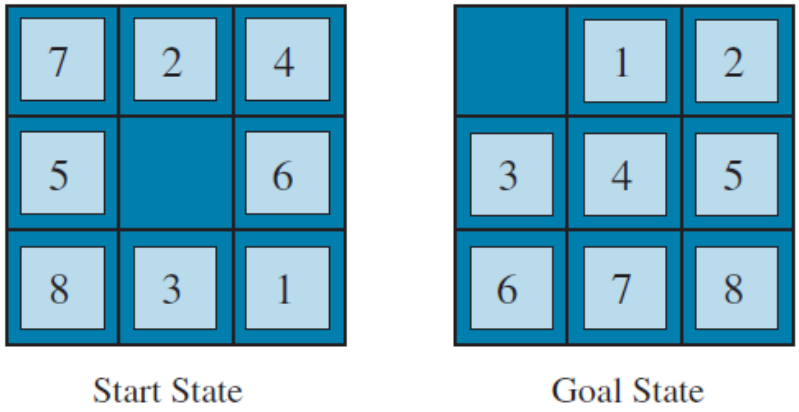
- h_1 = # tiles misplaced

  Admissible: any tile out of place will require ≥ 1 move to fix.

- h_2 = sum of Manhattan distances of the tiles from their goal positions

  Admissible: every move reduces the Manhattan distance of only one tile by ≤ 1.

Two Heuristics



Heuristics are needed for searching the vast state space.

♦ h_1 = # tiles misplaced

Admissible: any tile out of place will require ≥ 1 move to fix.

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

Tile	1	2	3	4	5	6	7	8
Manhattan distance	3	1	2	2	2	3	3	2

⇩

$h_2 = 18$

♦ h_2 = sum of Manhattan distances of the tiles from their goal positions

Admissible: every move reduces the Manhattan distance of only one tile by ≤ 1.

Neither heuristic overestimates the shortest solution (26 actions for the problem instance).

Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$
$$= \left(1 - (b^*)^{d+1}\right)/(1 - b^*)$$

Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$
$$= \left(1 - (b^*)^{d+1}\right)/(1 - b^*)$$

$$\Longrightarrow \quad (b^*)^{d+1} - (N+1)b^* + N = 0$$

Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$
$$= \left(1 - (b^*)^{d+1}\right)/(1 - b^*)$$

$$\Longrightarrow \quad (b^*)^{d+1} - (N+1)b^* + N = 0$$

(polynomial root finding – Com S 4770/5770)

Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

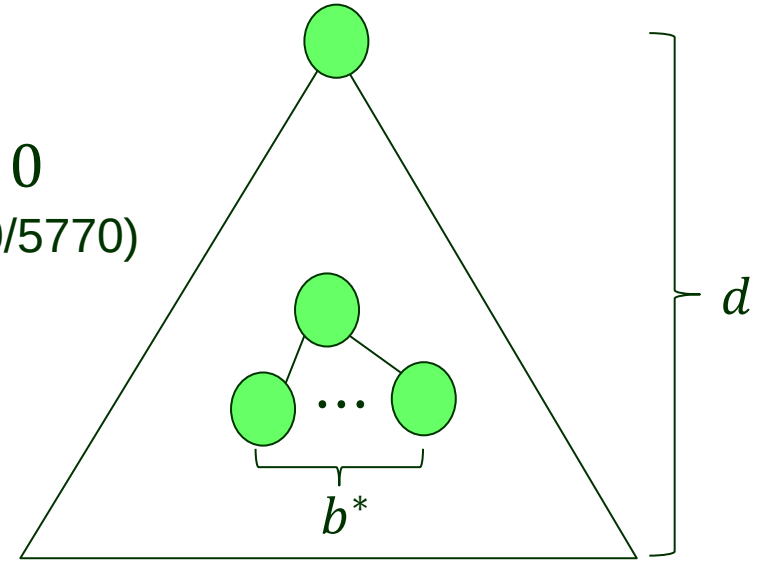
If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$
$$= \left(1 - (b^*)^{d+1}\right)/(1 - b^*)$$

$$\Longrightarrow \quad (b^*)^{d+1} - (N+1)b^* + N = 0$$

(polynomial root finding – Com S 4770/5770)

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height d in which every node at depth $< d$ has exactly b^*children.



Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

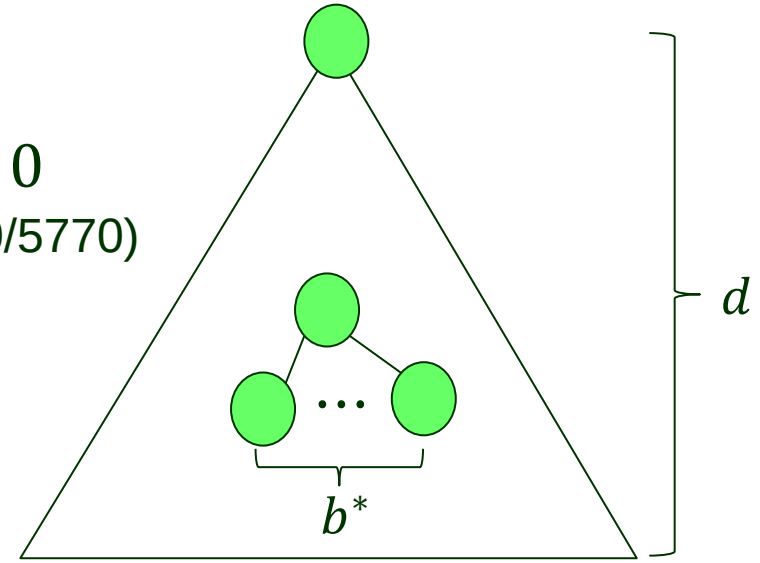
If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$
$$= \left(1 - (b^*)^{d+1}\right)/(1 - b^*)$$

$$\Longrightarrow \quad (b^*)^{d+1} - (N+1)b^* + N = 0$$

(polynomial root finding – Com S 4770/5770)

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height d in which every node at depth $< d$ has exactly b^*children.



e.g. A* finds a solution at depth 5 using 52 nodes has $b^* = 1.92$.

Performance Comparison on 8-Puzzle

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Figure 3.26 Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A* with h_1 (misplaced tiles), and A* with h_2 (Manhattan distance). Data are averaged over 100 puzzles for each solution length d from 6 to 28.