Heuristic Functions

Outline

I. Properties of Heuristics

II. Variations of A* search

III. Generating heuristics

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

I. Admissible Heuristic

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

I. Admissible Heuristic

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

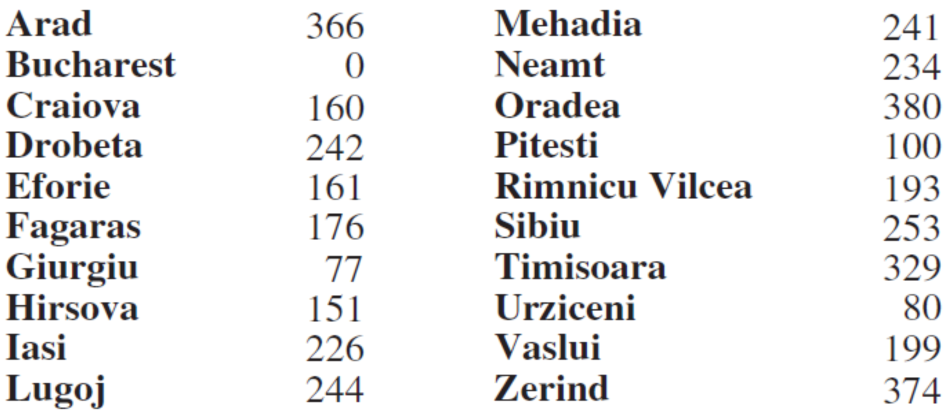
A heuristic is *admissible* if it *never overestimates* the cost to reach a goal.

I. Admissible Heuristic

- 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



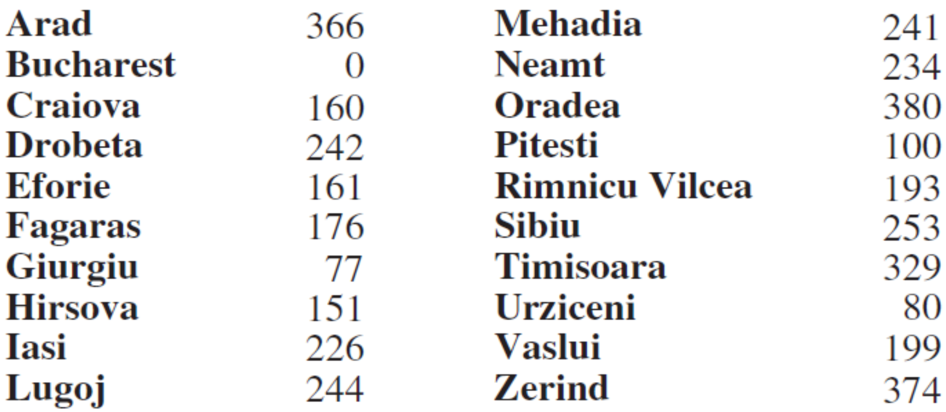
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

I. Admissible Heuristic

- 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^*.

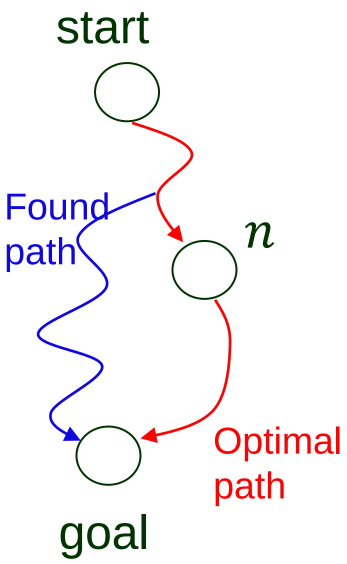
Optimality of A*

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^*.

$\Downarrow$

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



Optimality of A*

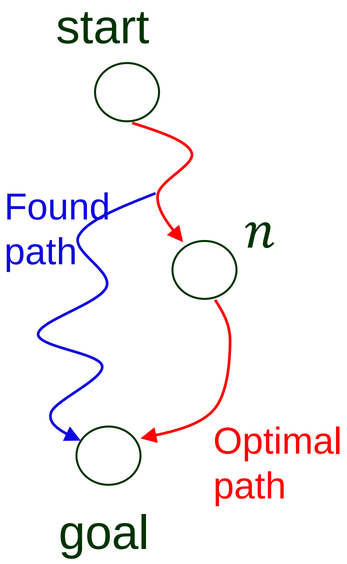
Theorem A* is cost-optimal with an admissible heuristic.

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

$\Downarrow$

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

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

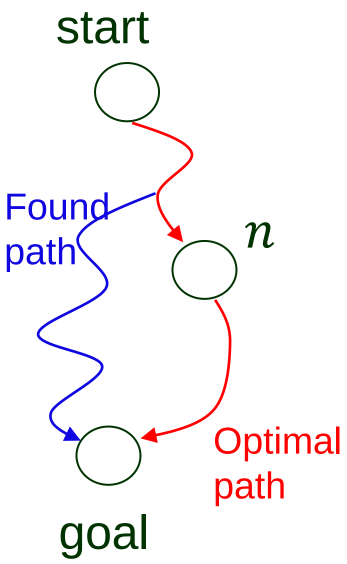


Optimality of A*

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

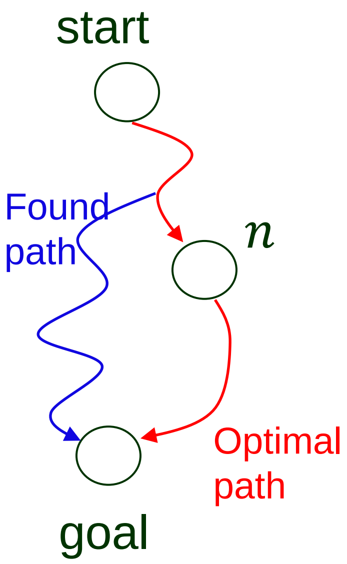
$h^*(n)$: optimal cost from n to a goal

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^*.

$$\Downarrow$$



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

$$\Downarrow$$

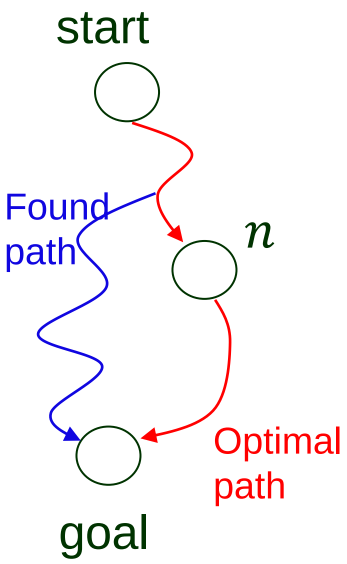
$f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so n would have been expanded)

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)

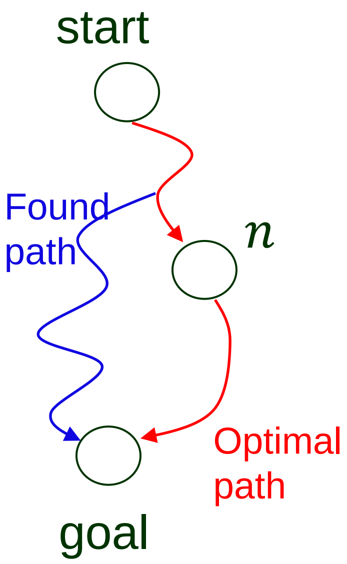
But $f(n) = g(n) + h(n)$

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^*.

$$\Downarrow$$



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

$$\Downarrow$$

$f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so n would have been expanded)

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

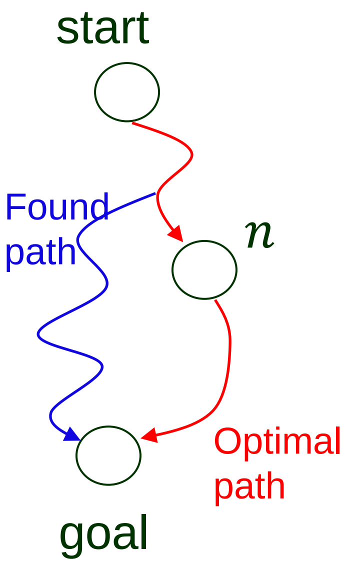
$= g^*(n) + h(n)$ (n on the optimal path)

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 $f(n) \le C^* < C$ so n would have been expanded)

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

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

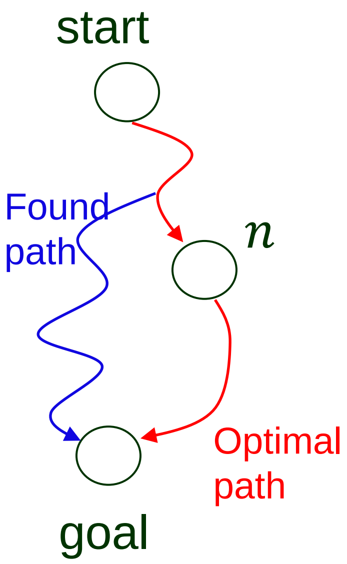
$\le g^*(n) + h^*(n)$ ($h(n) \le h^*(n)$ due to admissibility)

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^*.

$$\Downarrow$$



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

$$\Downarrow$$

$f(n) > C^*$ (otherwise $f(n) \le C^* < C$ so n would have been expanded)

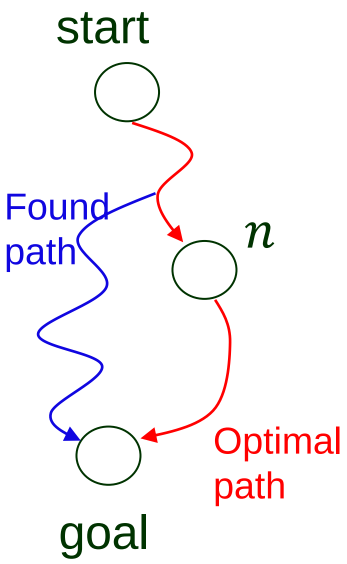
But $$\begin{aligned} f(n) &= g(n) + h(n) \\ &= g^*(n) + h(n) && (n \text{ on the optimal path}) \\ &\le g^*(n) + h^*(n) && (h(n) \le h^*(n) \text{ due to admissibility}) \\ &= C^* \end{aligned}$$

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)

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)

$= C^*$

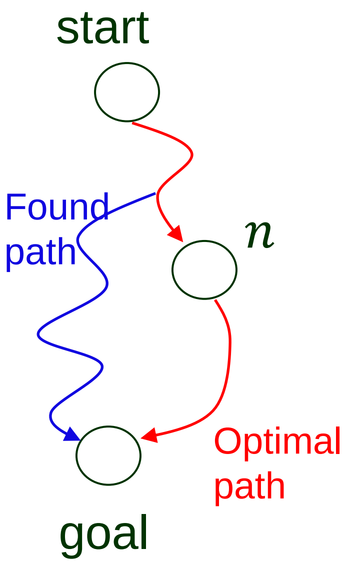
That is, $f(n) \leq C^*$, contradicting with $f(n) > C^*$.

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 $f(n) \le C^* < C$ so n would have been expanded)

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

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

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

$= C^*$

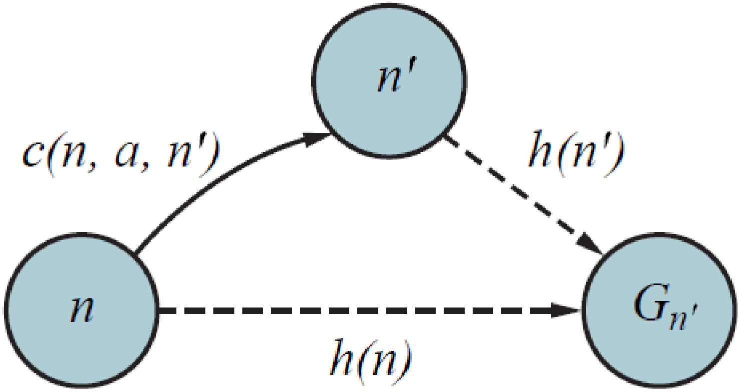
That is, $f(n) \le C^*$, contradicting with $f(n) > C^*$. ☺

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*)

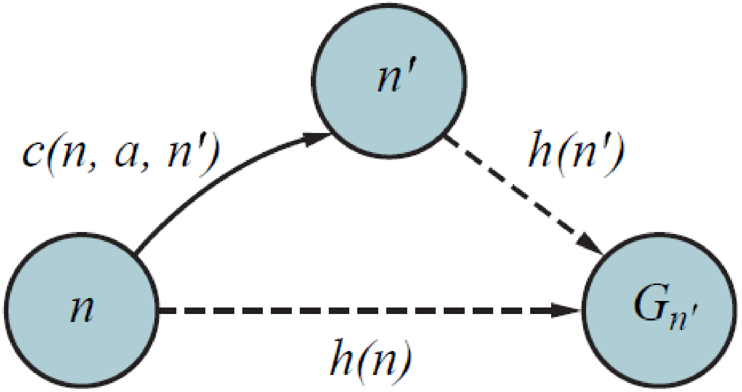


Consistent Heuristic

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*)



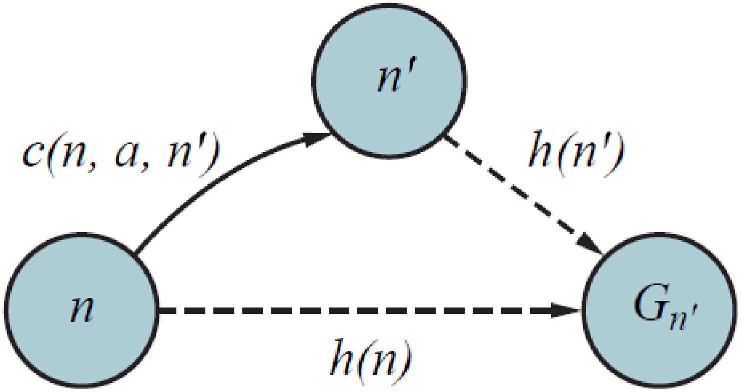
- Every consistent heuristic is admissible.

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

What If the Heuristic Is Inadmissible?

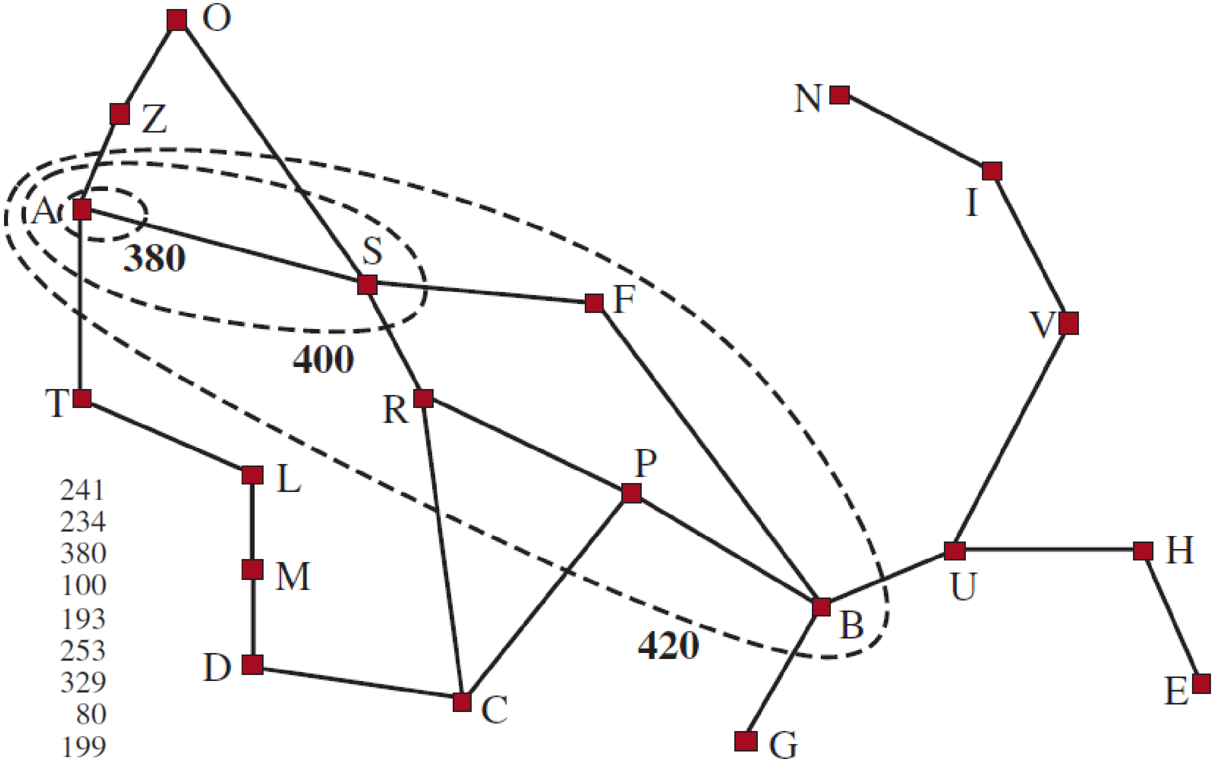
In such a situation, A* may or may not be cost-optimal.

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

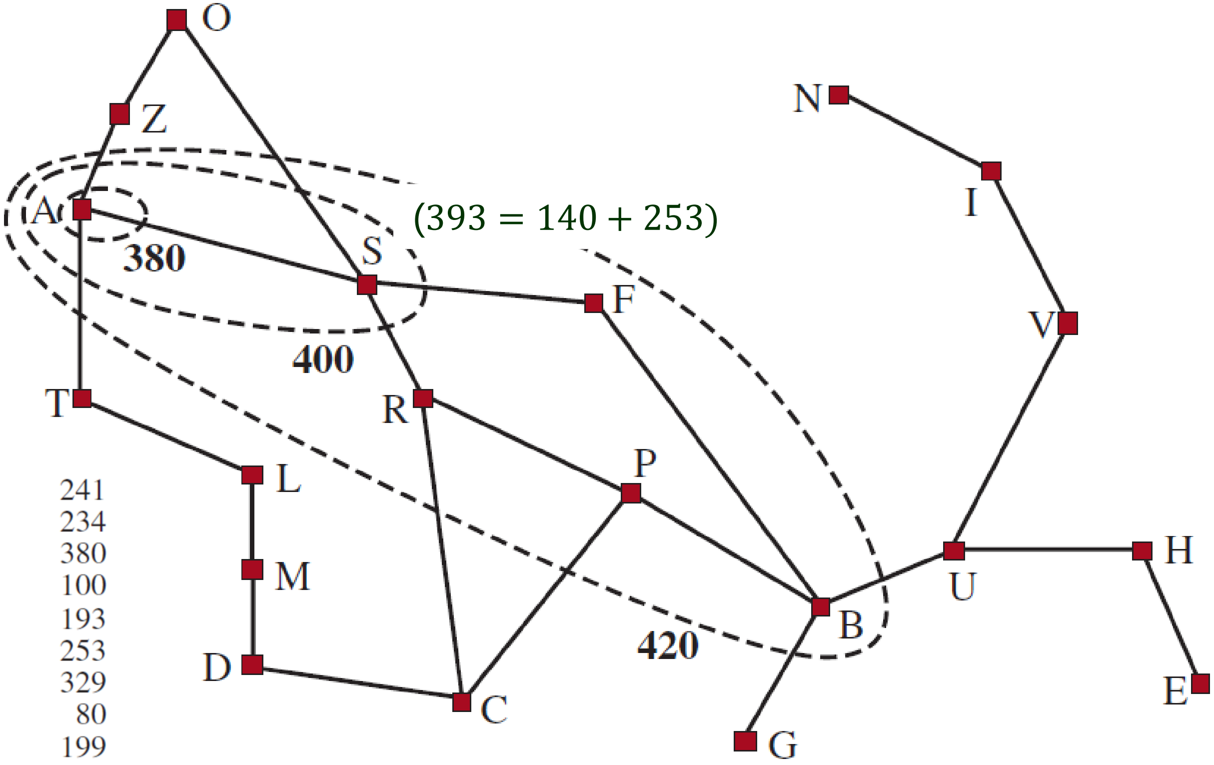
A *contour* labeled by a cost c encloses all the nodes n with $f(n) = g(n) + h(n) \leq c$.



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 $f(n) = g(n) + h(n) \leq c$.



$(393 = 140 + 253)$

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

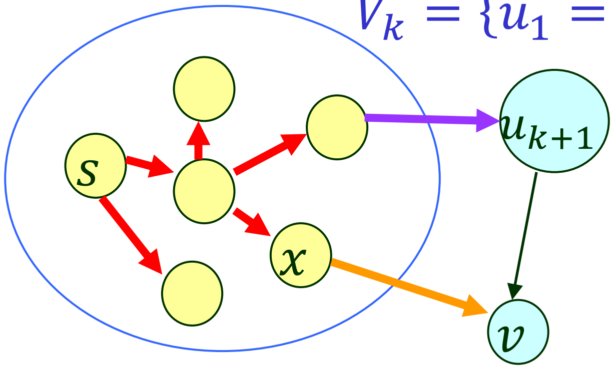
Some Search Contours

- 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.

$V_k = \{u_1 = s, \ldots, u_k\}$

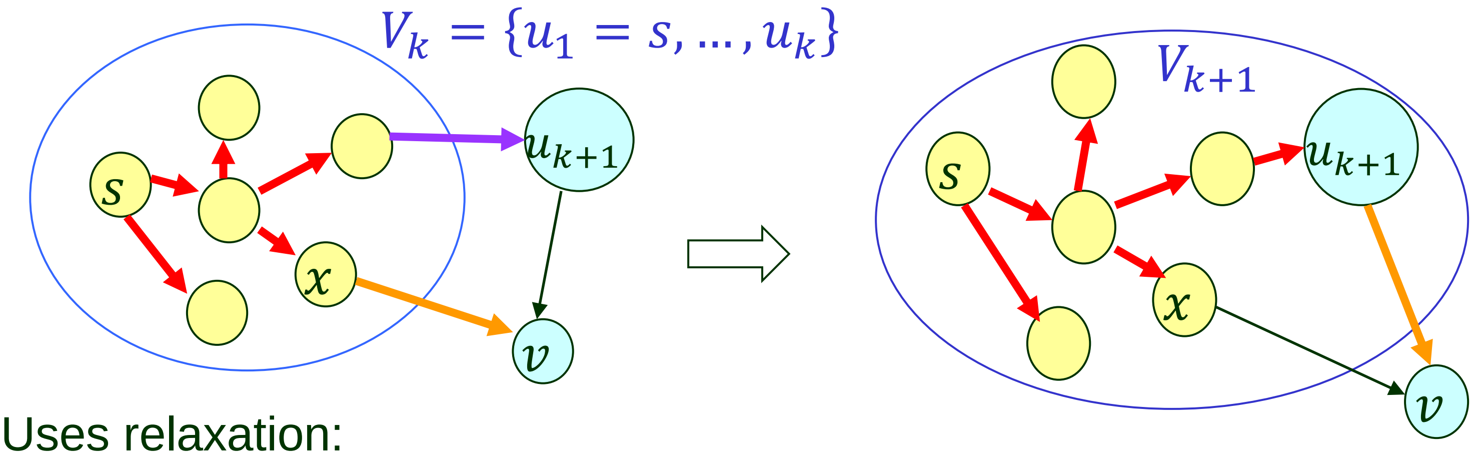


Uses relaxation:

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

Some Search Contours

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

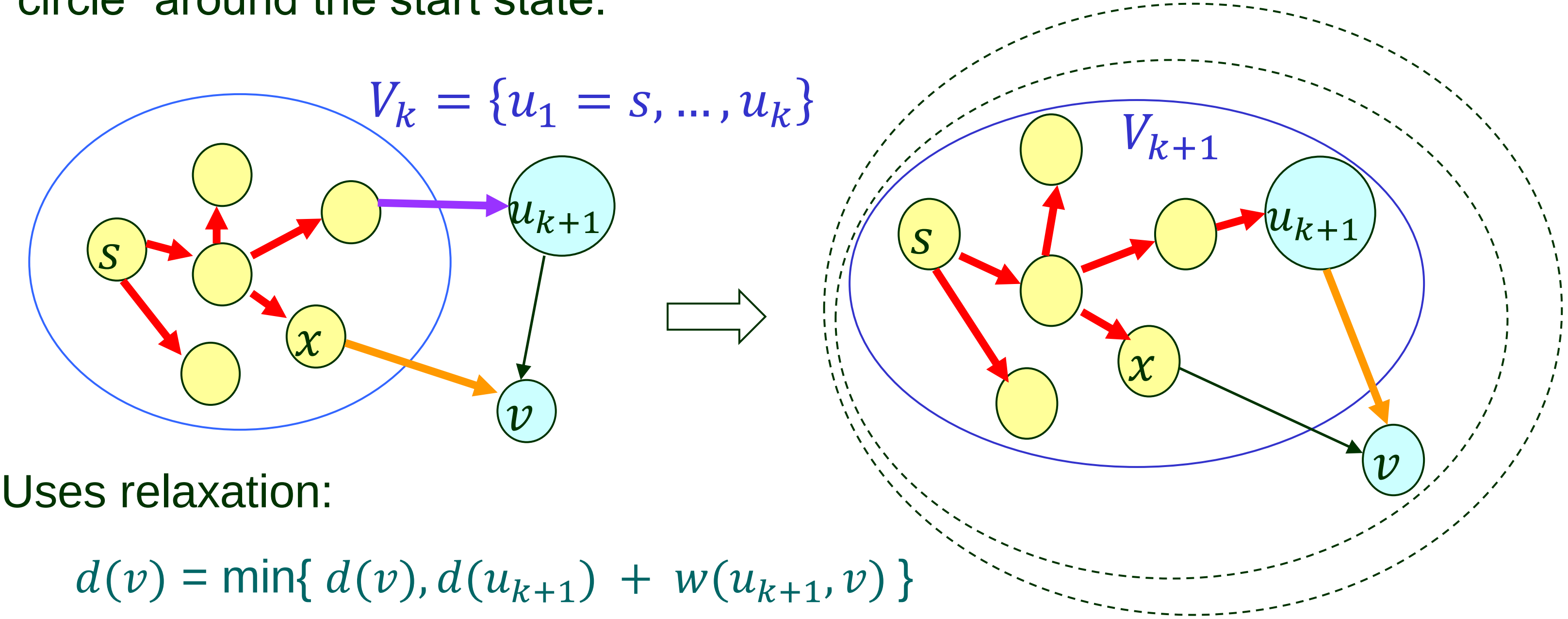


Uses relaxation:

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

Some Search Contours

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



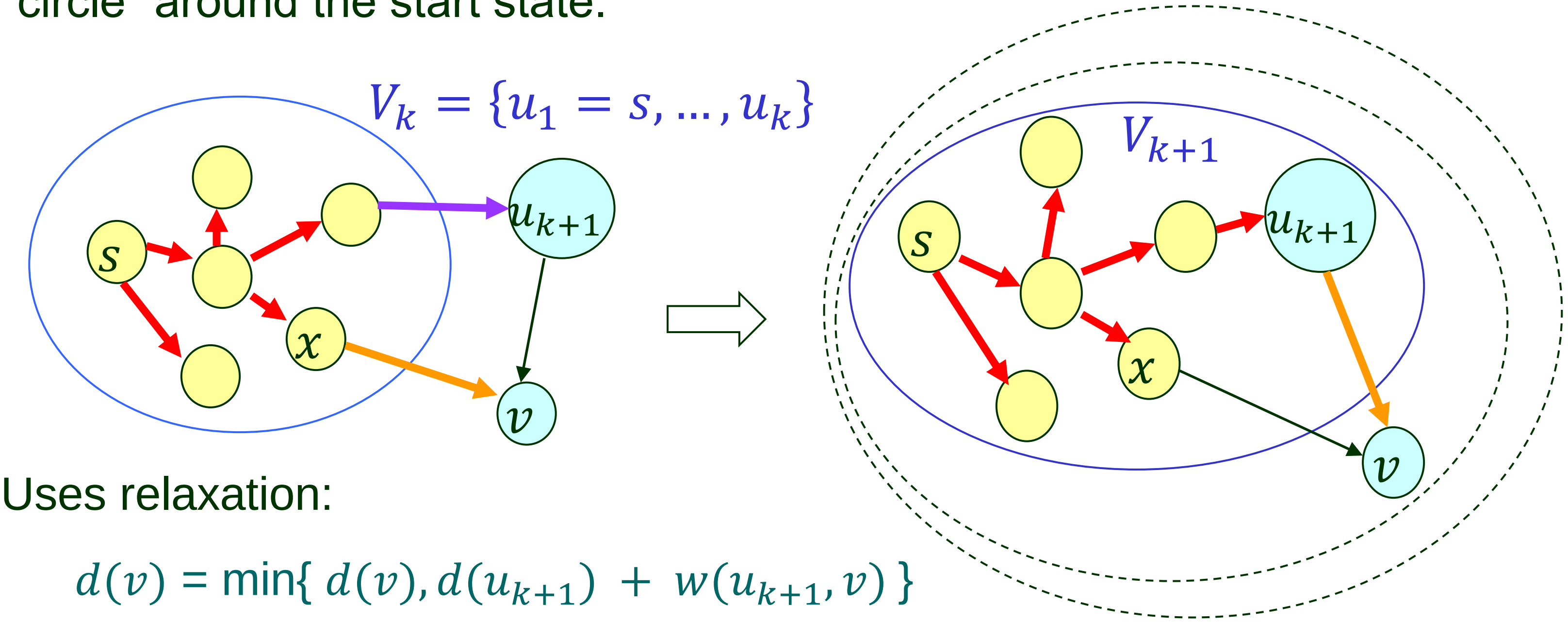
$$V_k = \{u_1 = s, \dots, u_k\}$$

Uses relaxation:

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

Some Search Contours

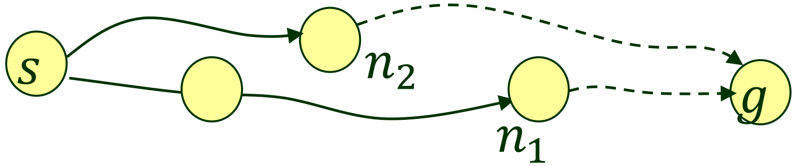
- Dijkstra’s algorithm (i.e., uniform search) would have contours of g-cost to “circle” around the start state.



Uses relaxation:

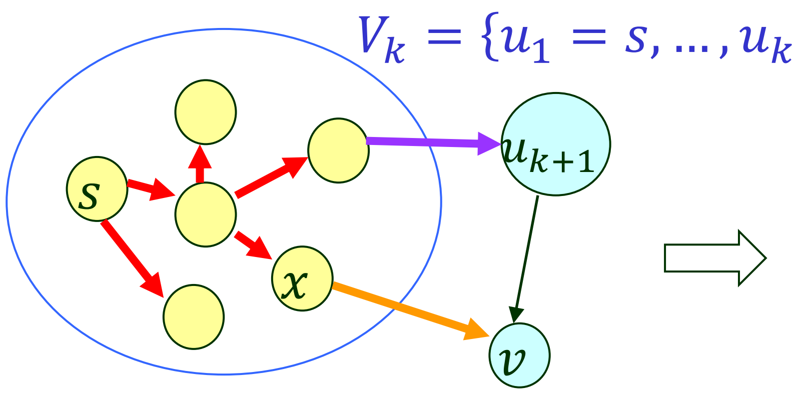
$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

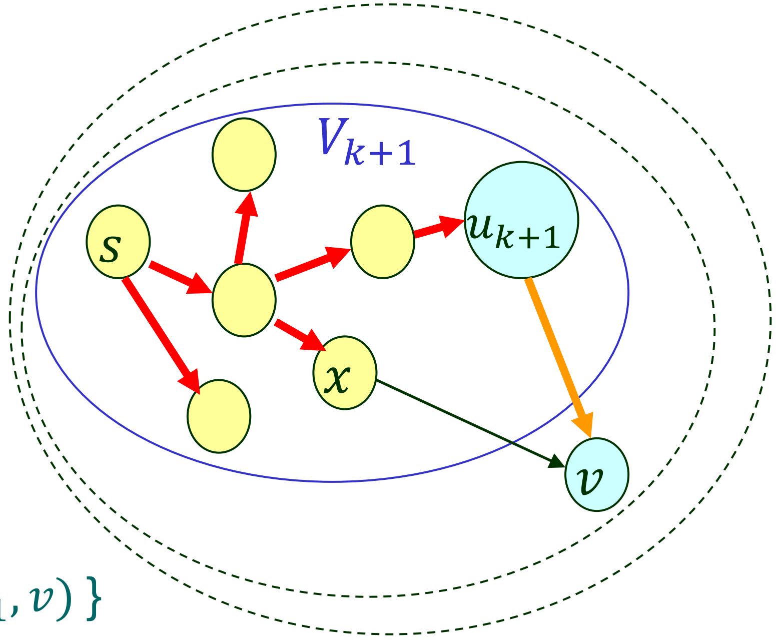
- With a good h, the $g + h$ bands will stretch toward a goal state.



Some Search Contours

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

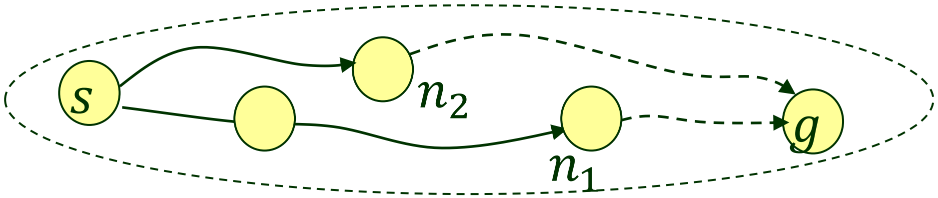




Uses relaxation:

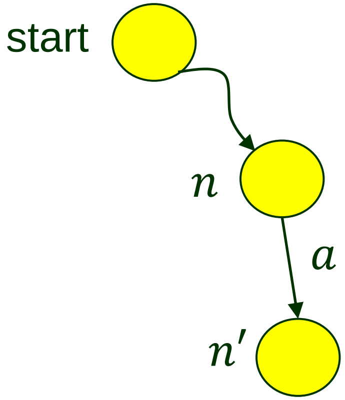
$$d(v) = \min\{\, d(v), d(u_{k+1}) + w(u_{k+1}, v) \,\}$$

- 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.

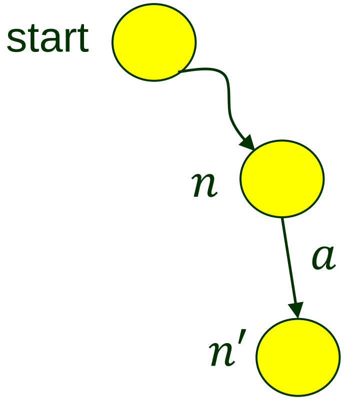


Cost at n: $g(n) + h(n)$

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

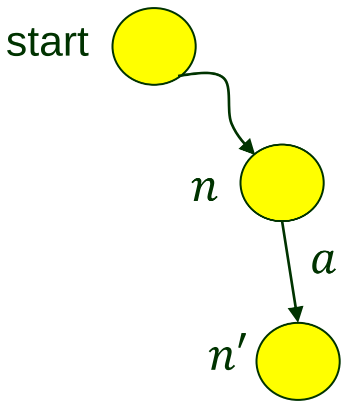
Cost at its successor n': $\underbrace{g(n) + c(n,a,n')}_{= g(n')} + h(n')$

The path's cost increases *monotonically* iff

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

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

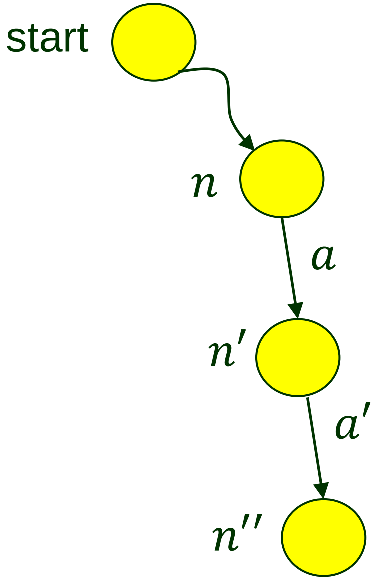
$$g(n) + h(n) \le g(n) + c(n, a, n') + h(n')$$

$$\Updownarrow$$

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

(consistent heuristic)

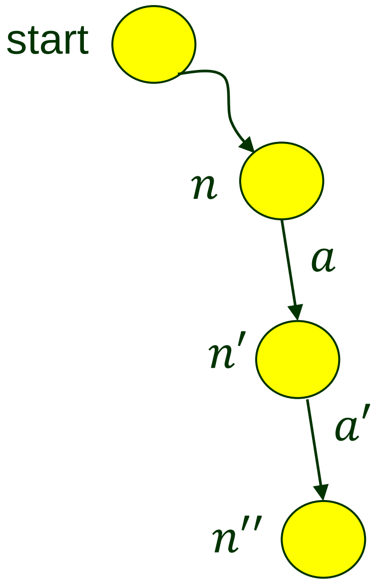
Consecutive Nodes Scored the Same



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

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

Consecutive Nodes Scored the Same



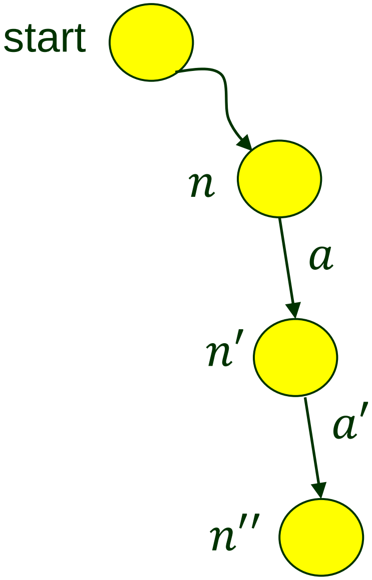
$$g(n) + h(n) = \overbrace{g(n) + c(n, a, n')}^{= g(n')} + h(n')$$

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

$$\Updownarrow$$

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

Consecutive Nodes 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. Sacrificing Search

- A* explores a lot of nodes due to equal weighting of g and h in $f = g + h$ which often distracts it from the optimal path.
- Can explore fewer nodes if we are okay with satisficing (suboptimal but “good enough”) solutions .

II. Sacrificing Search

- A* explores a lot of nodes due to equal weighting of g and h in $f = g + h$ which often distracts it from the optimal path.
- Can explore fewer nodes if we are okay with satisficing (suboptimal but "good enough") solutions .

Use an inadmissible heuristic.

II. Sacrificing Search

- A* explores a lot of nodes due to equal weighting of g and h in $f = g + h$ which often distracts it from the optimal path.
- Can explore fewer nodes if we are okay with satisficing (suboptimal but “good enough”) solutions .

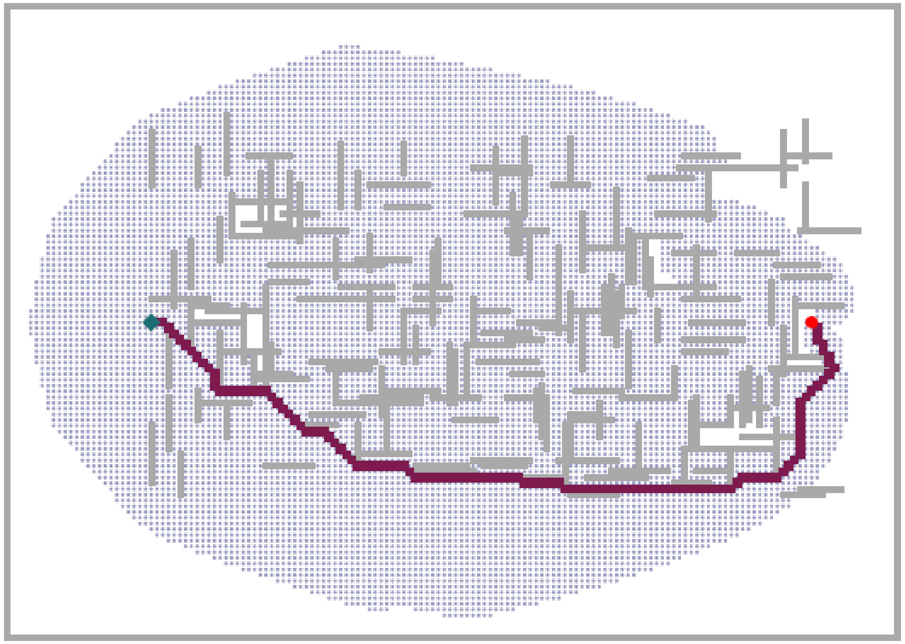
Use an inadmissible heuristic.

Idea of *detour index*: multiplier applied to the straight-line distance.

e.g., a detour index of 1.3 implies a good estimate of 13 miles between two cities that are 10 miles apart.

Weighted A*

Evaluation function: $f(n) = g(n) + W \times h(n)$ for some $W > 1$.

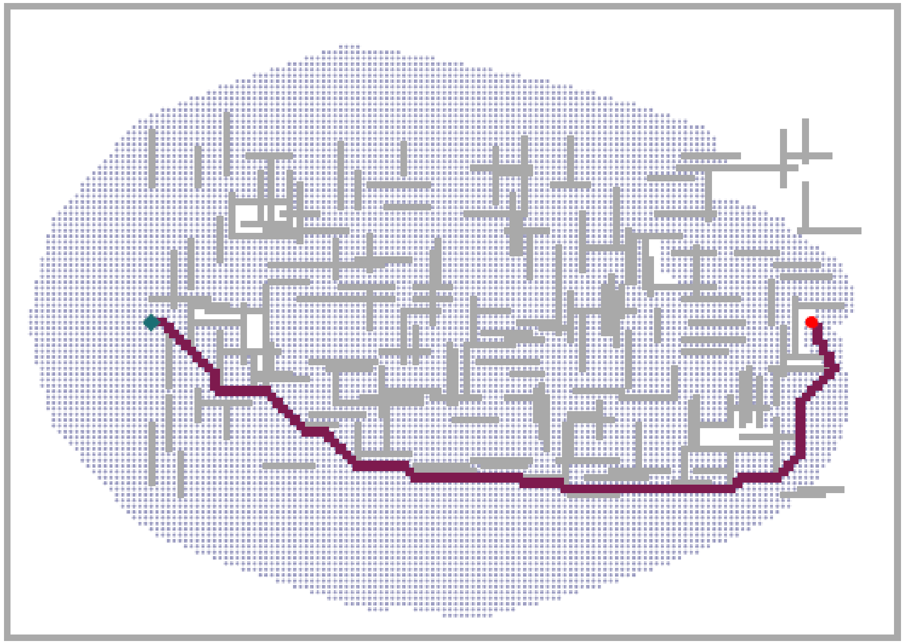


A* search

Gray bars: obstacles
Dots: reached states

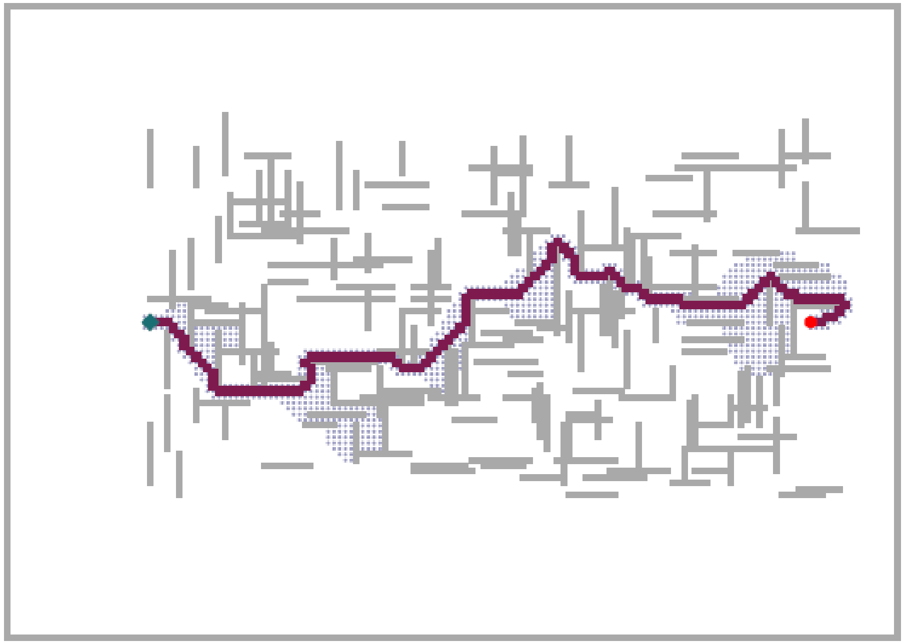
Weighted A*

Evaluation function: $f(n) = g(n) + W \times h(n)$ for some $W > 1$.



A* search

Gray bars: obstacles
Dots: reached states



Weighted A* search
($W = 2$ on the same grid)

Weighted A* As a Generalization

- Weighted A* finds a solution with cost between C^* and $W \times C^*$.
- Cost is usually much closer to C^* in practice.

Weighted A* search $g(n) + W \times h(n) \quad (1 \leq W < \infty)$

Weighted A* As a Generalization

- Weighted A* finds a solution with cost between C^* and $W \times C^*$.
- Cost is usually much closer to C^* in practice.

A* search	$g(n) + h(n)$	$(W = 1)$
Uniform-cost search	$g(n)$	$(W = 0)$
Best-cost search	$h(n)$	$(W = \infty)$
Weighted A* search	$g(n) + W \times h(n)$	$(1 \le W < \infty)$

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.
- Iterative-deepening A* search (IDA*)
  - The cutoff at each iteration is the f-cost.

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
    - ♦ Less memory and faster execution.
    - ♣ Incomplete and suboptimal.

- Iterative-deepening A* search (IDA*)
    - The cutoff at each iteration is the f-cost.

    $S = \{n \mid n$ generated in the previous iteration and $f(n) >$ old $f_{\text{cutoff}}\}$

    (previous iteration)

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.

- Iterative-deepening A* search (IDA*)
  - The cutoff at each iteration is the f-cost.

$$S = \{n \mid n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}}\}$$

(previous iteration)

$$\text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n)$$

(current iteration)

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.
- Iterative-deepening A* search (IDA*)
  - The cutoff at each iteration is the f-cost.

$$S = \{n \mid n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}}\}$$

(previous iteration)

$$\text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n)$$

(current iteration)

  - ♦ Steady progress towards the goal if f-cost of every path is an integer.

$\leq C^*$ iterations

Memory-Bounded Search

- Beam search keeps the k nodes with the best f scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.

- Iterative-deepening A* search (IDA*)
  - The cutoff at each iteration is the f-cost.

$S = \{n \mid n$ generated in the previous iteration and $f(n) >$ old $f_{\text{cutoff}}\}$

(previous iteration)

$$\text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n)$$

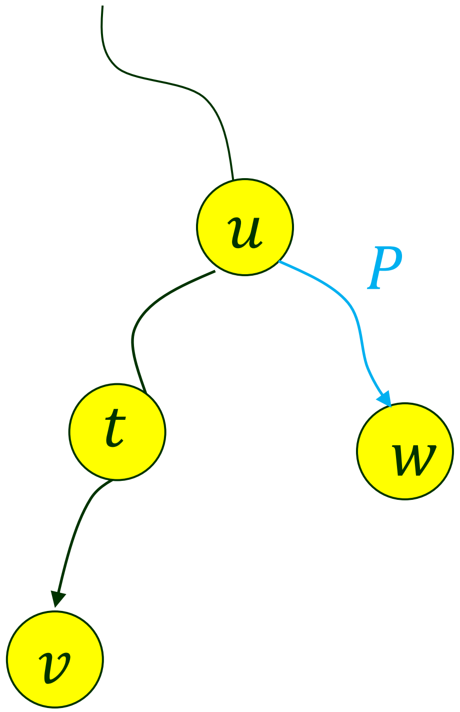
(current iteration)

  - ♦ Steady progress towards the goal if f-cost of every path is an integer.

$\leq C^*$ iterations

  - ♣ In the worst case, #iterations = #states

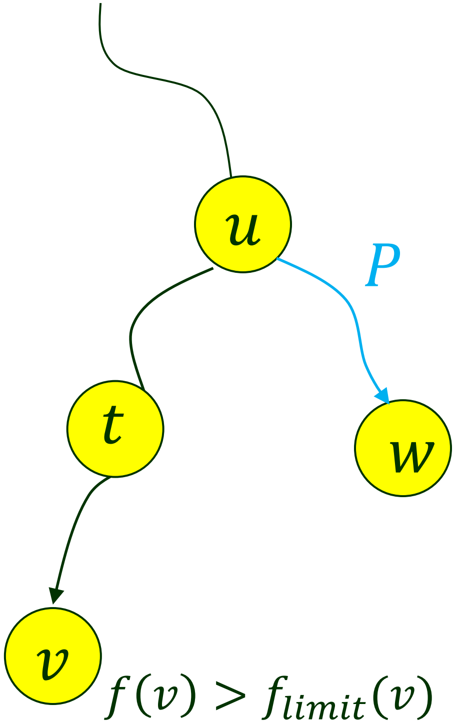
Recursive Best-First Search (RBFS)



$f_{limit}(v)$: f-value of the *best alternative path* from any *ancestor* of the node v.

- Best-first search if at the currently visited node v it holds that $f(v) \leq f_{limit}(v)$.

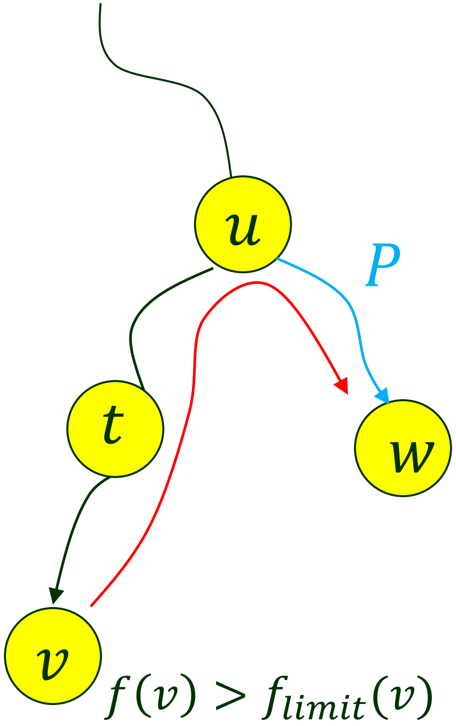
Recursive Best-First Search (RBFS)



$f_{limit}(v)$: f-value of the *best alternative path* from any *ancestor* of the node v.

- Best-first search if at the currently visited node v it holds that $f(v) \leq f_{limit}(v)$.
- Otherwise (i.e., $f(v) > f_{limit}(v)$),

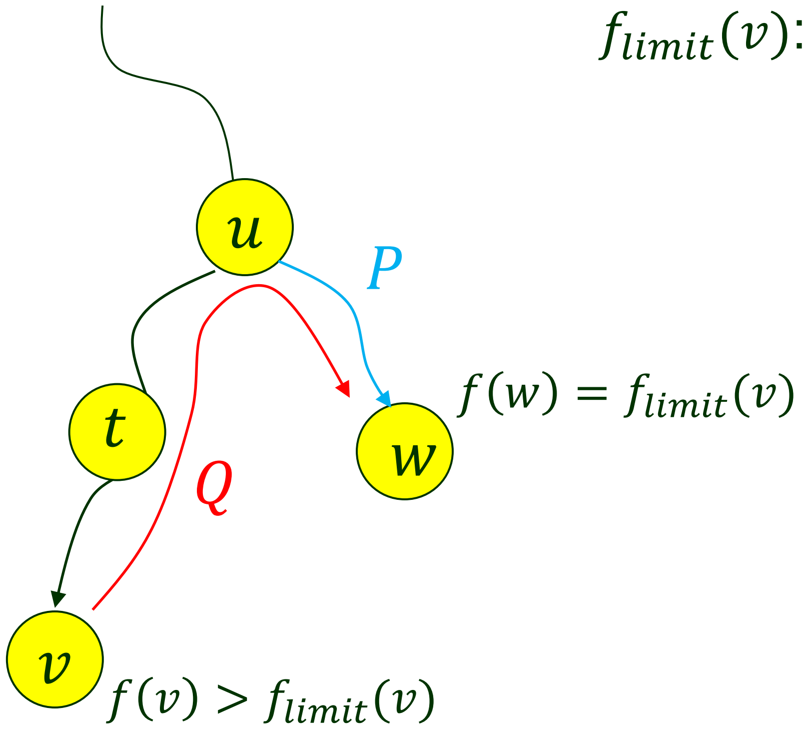
Recursive Best-First Search (RBFS)



$f_{limit}(v)$: f-value of the *best alternative path* from any *ancestor* of the node v.

- Best-first search if at the currently visited node v it holds that $f(v) \leq f_{limit}(v)$.
- Otherwise (i.e., $f(v) > f_{limit}(v)$),
  - unwinds back to the alternative path P;

Recursive Best-First Search (RBFS)

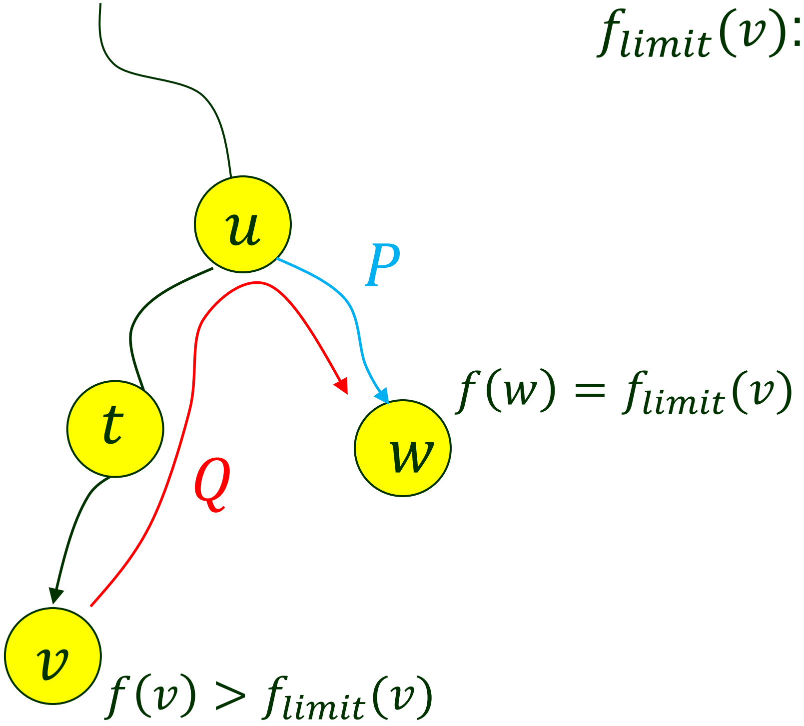


$f_{limit}(v)$: f-value of the *best alternative path* from any *ancestor* of the node v.

- Best-first search if at the currently visited node v it holds that $f(v) \leq f_{limit}(v)$.
- Otherwise (i.e., $f(v) > f_{limit}(v)$),
  - unwinds back to the alternative path P;

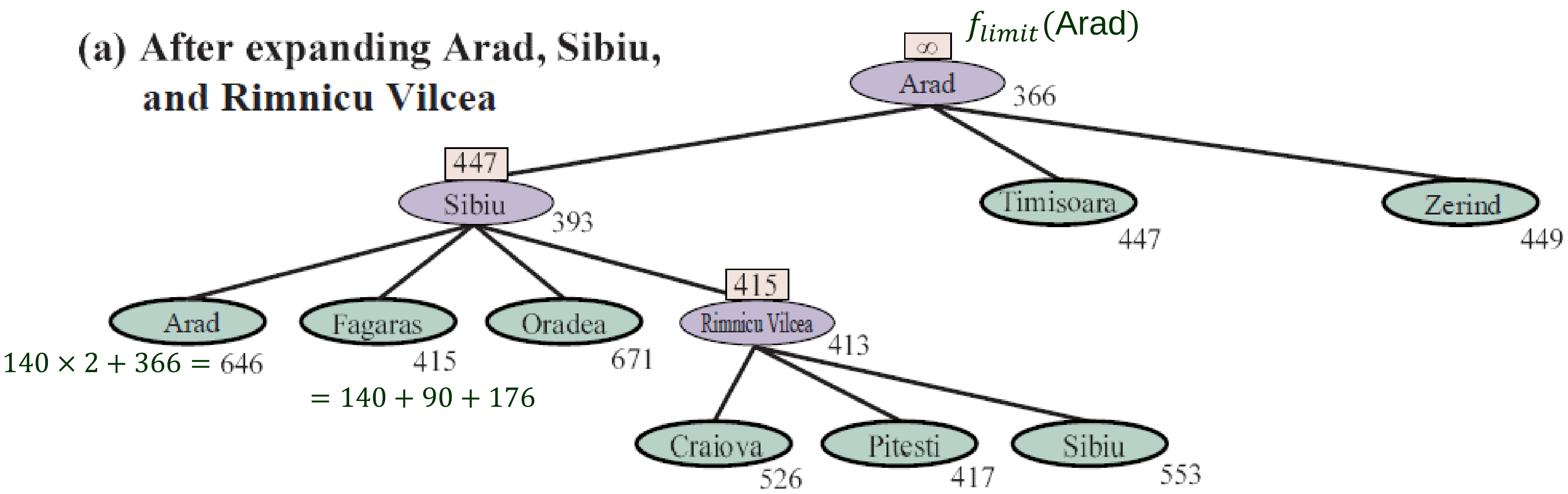
Recursive Best-First Search (RBFS)

$f_{limit}(v)$: f-value of the *best alternative path* from any *ancestor* of the node v.



- Best-first search if at the currently visited node v it holds that $f(v) \leq f_{limit}(v)$.
- Otherwise (i.e., $f(v) > f_{limit}(v)$),
  - unwinds back to the alternative path P;
  - update the f-value of every node along the path Q with the best f- value of its children.

RBFS Example



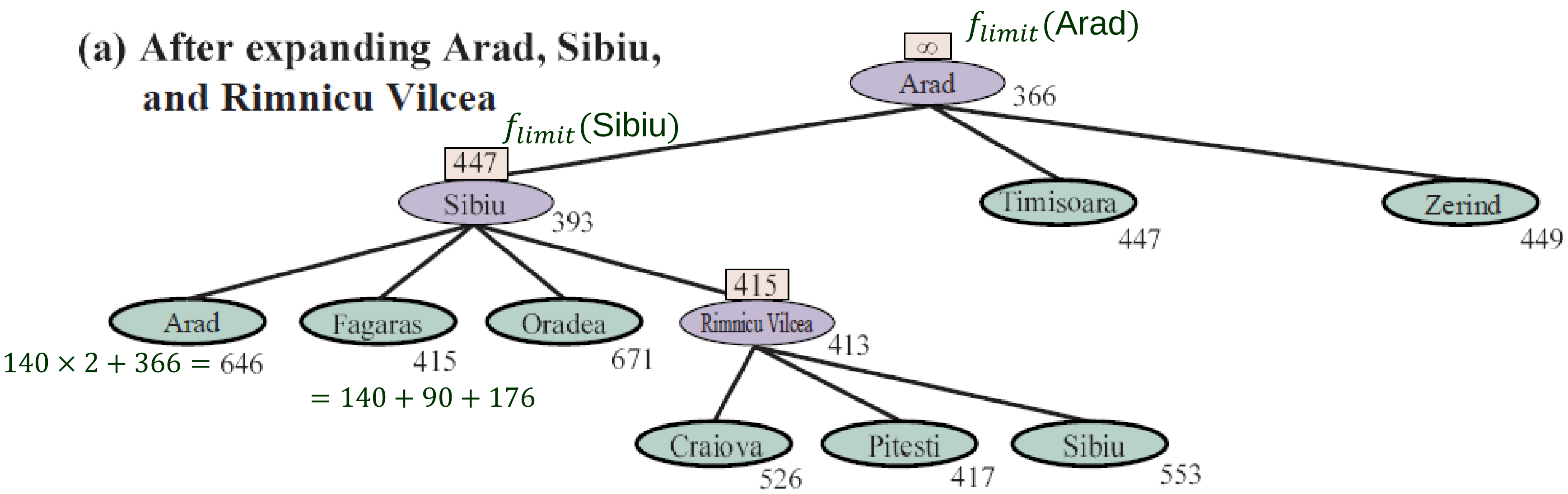
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

f_{limit}(Arad)

$140 \times 2 + 366 = 646$

$= 140 + 90 + 176$

RBFS Example



(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

f_{limit}(Arad)

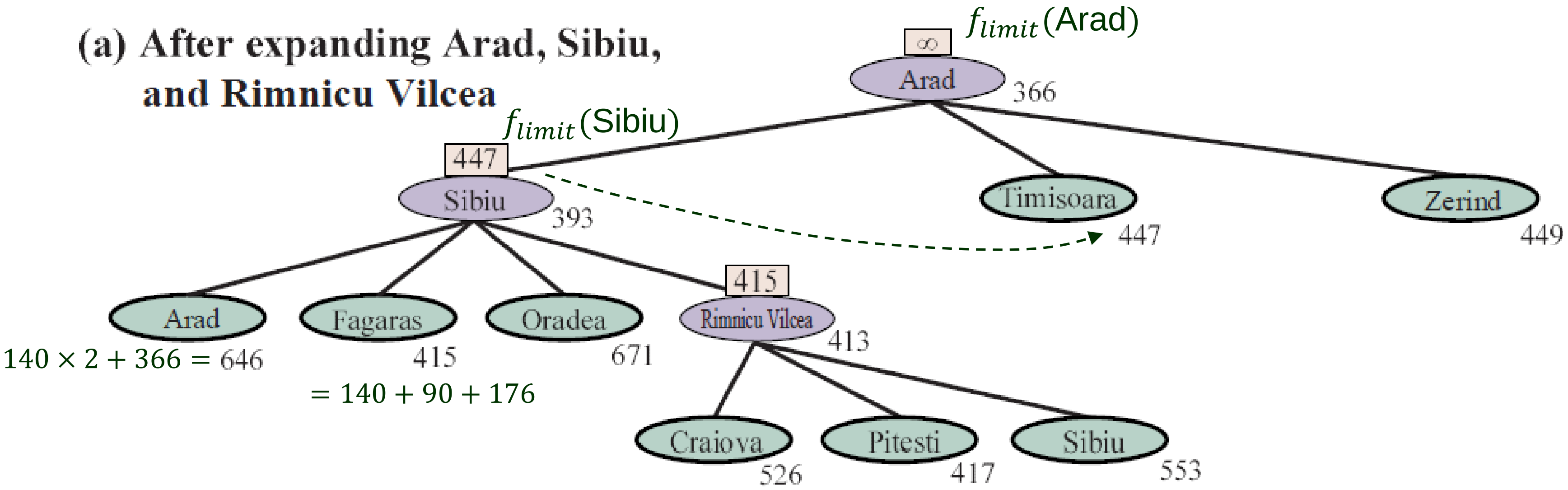
f_{limit}(Sibiu)

$140 \times 2 + 366 = 646$

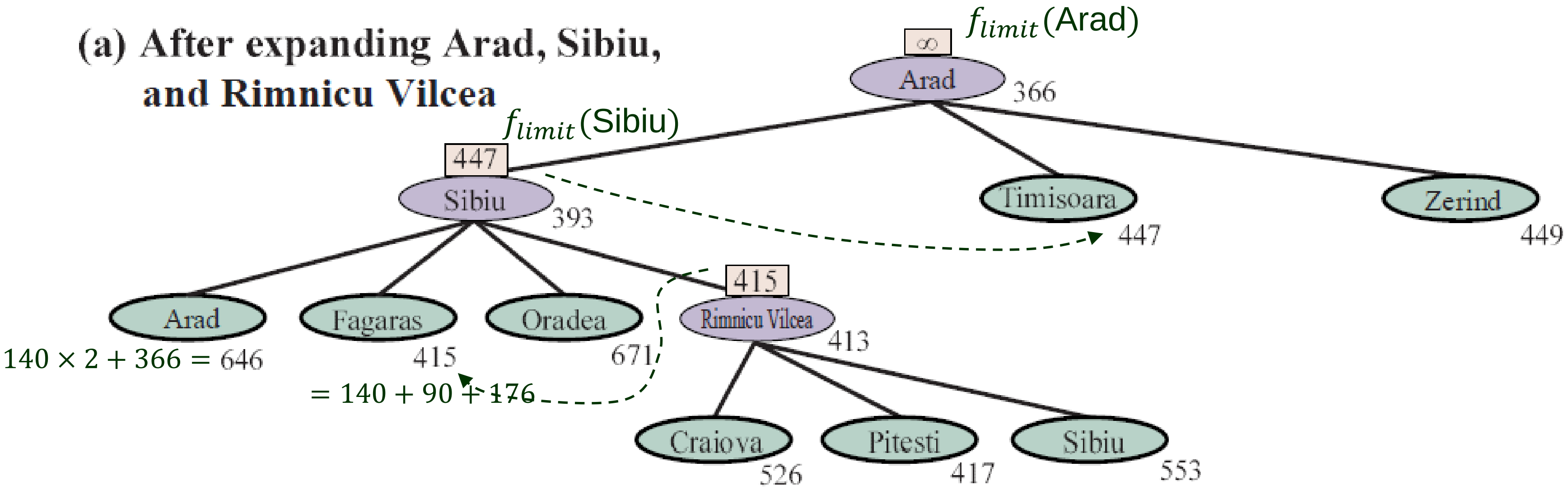
$= 140 + 90 + 176$

RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



RBFS Example



(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

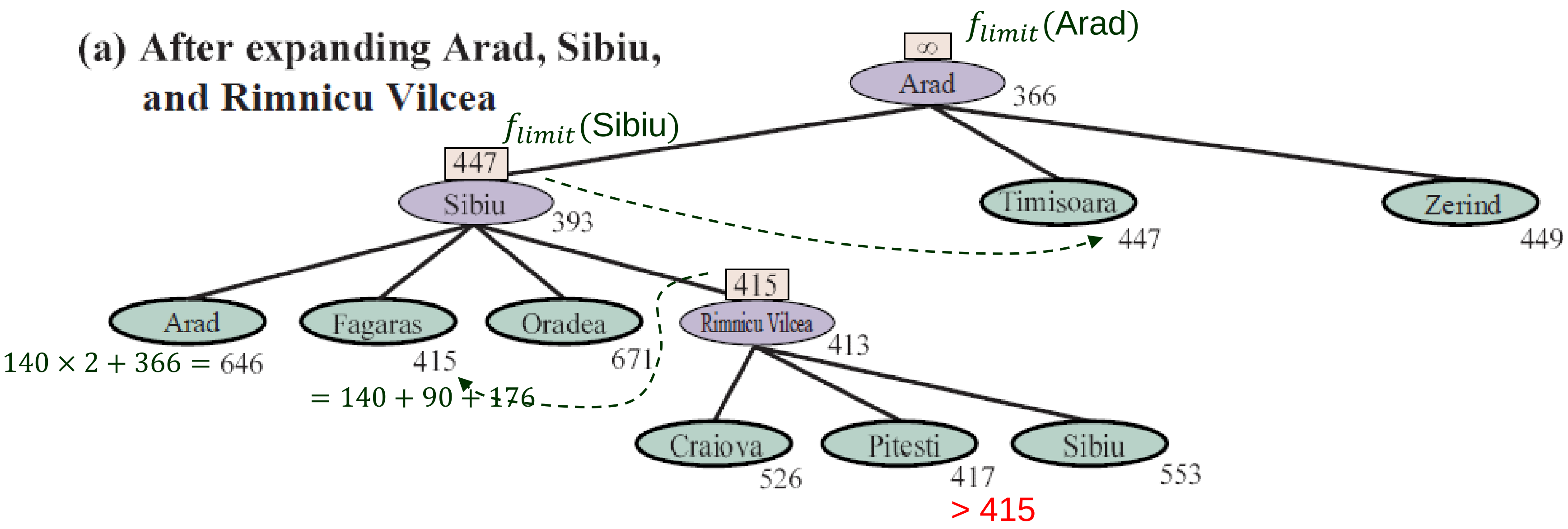
f_{limit}(Arad)

f_{limit}(Sibiu)

$140 \times 2 + 366 = 646$

$= 140 + 90 + 176$

RBFS Example



(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

f_{limit}(Arad)

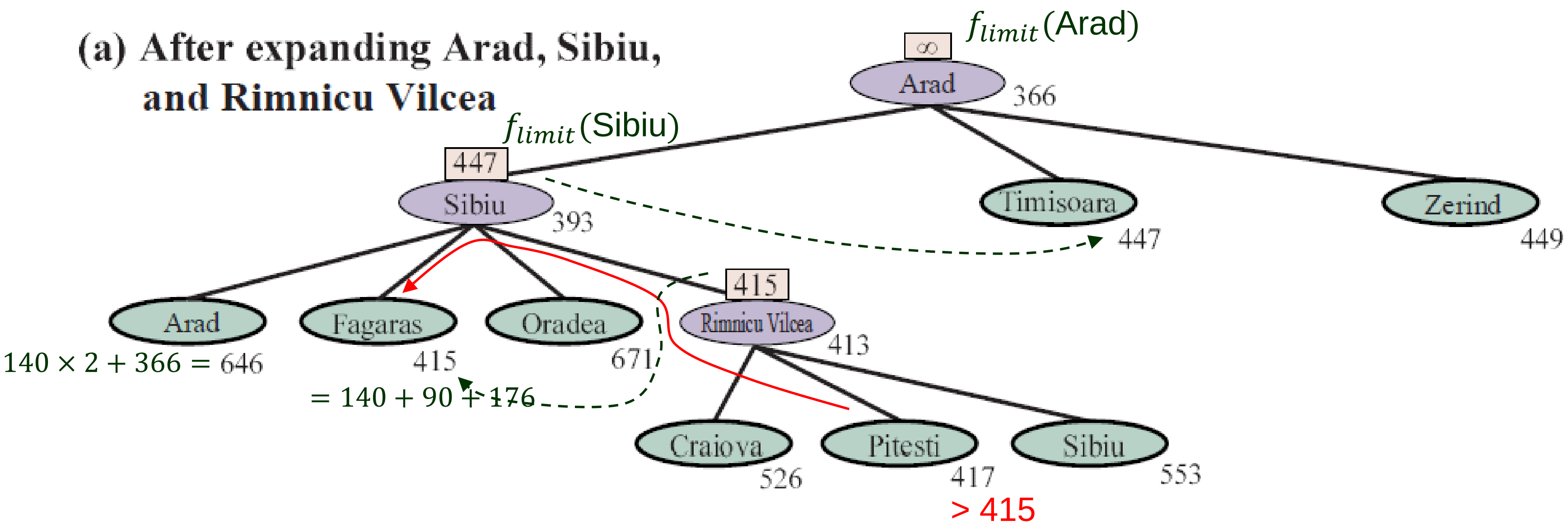
f_{limit}(Sibiu)

$140 \times 2 + 366 = 646$

$= 140 + 90 + 176$

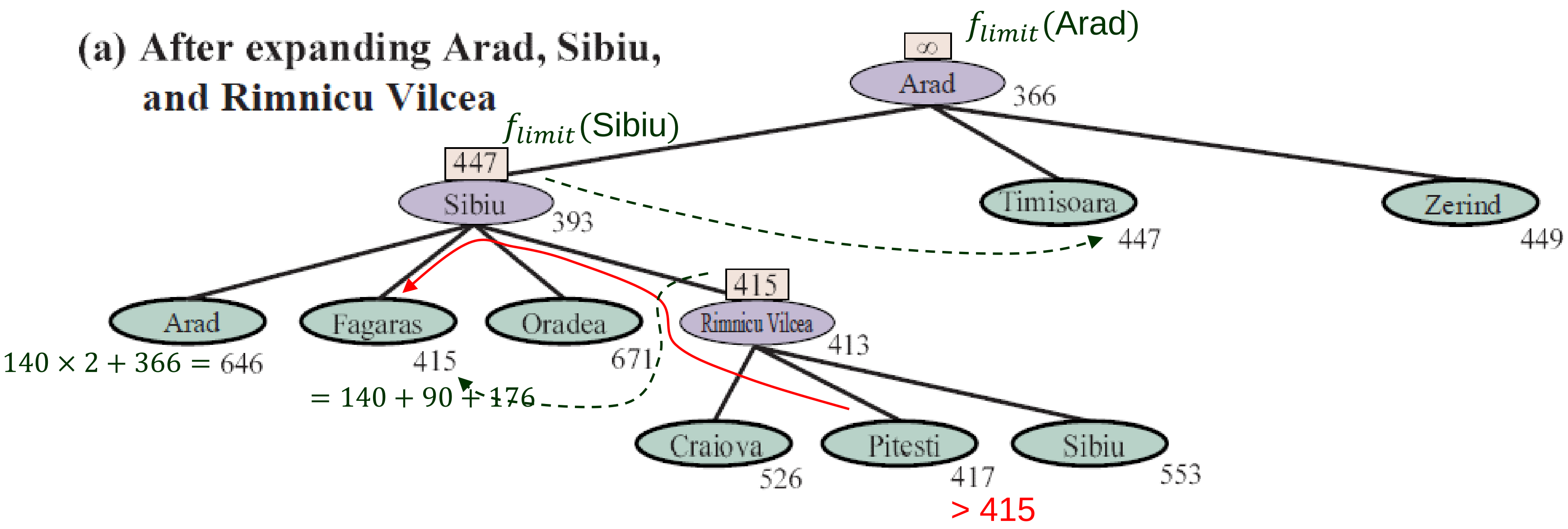
> 415

RBFS Example

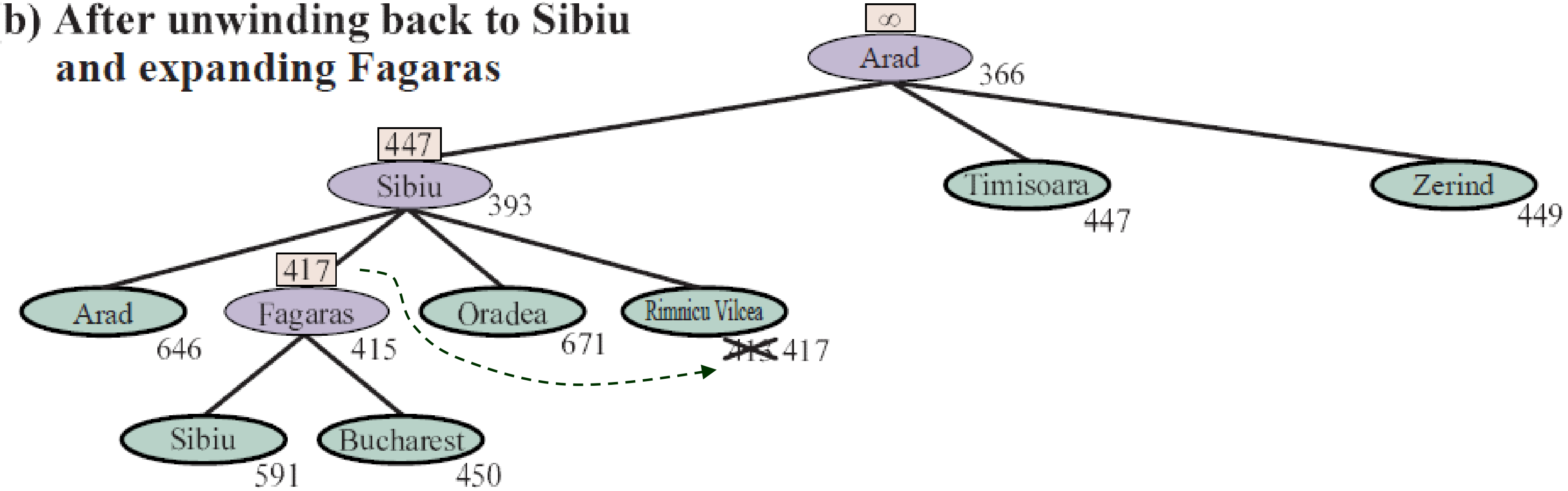

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

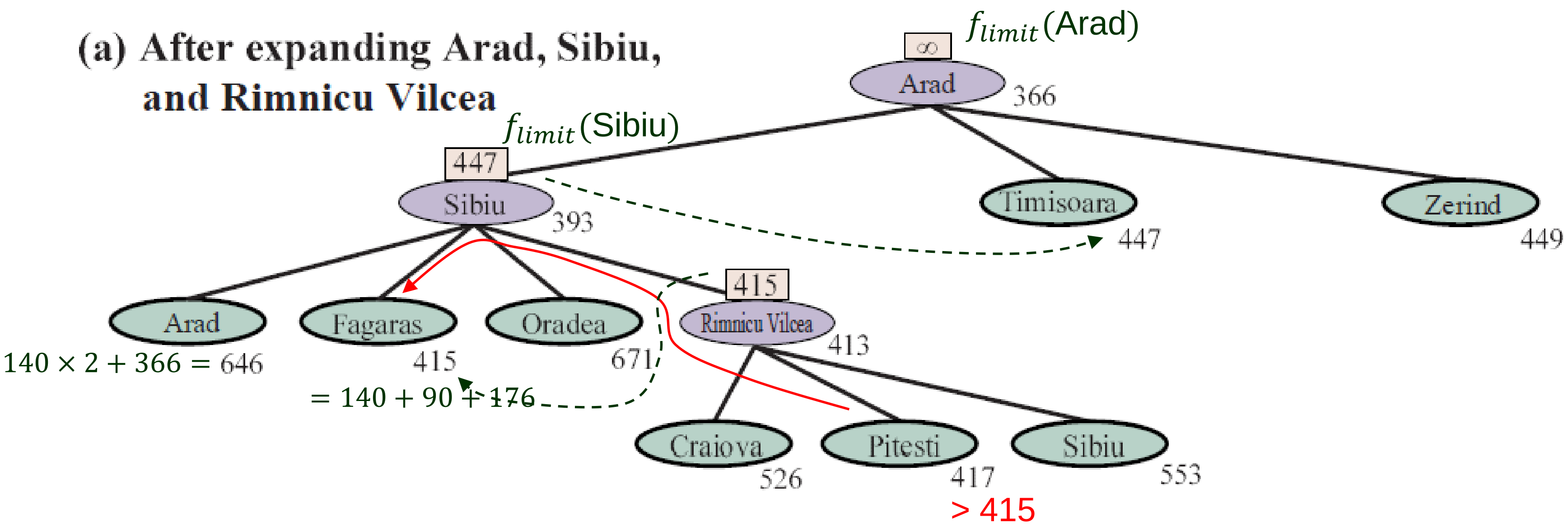


(b) After unwinding back to Sibiu and expanding Fagaras

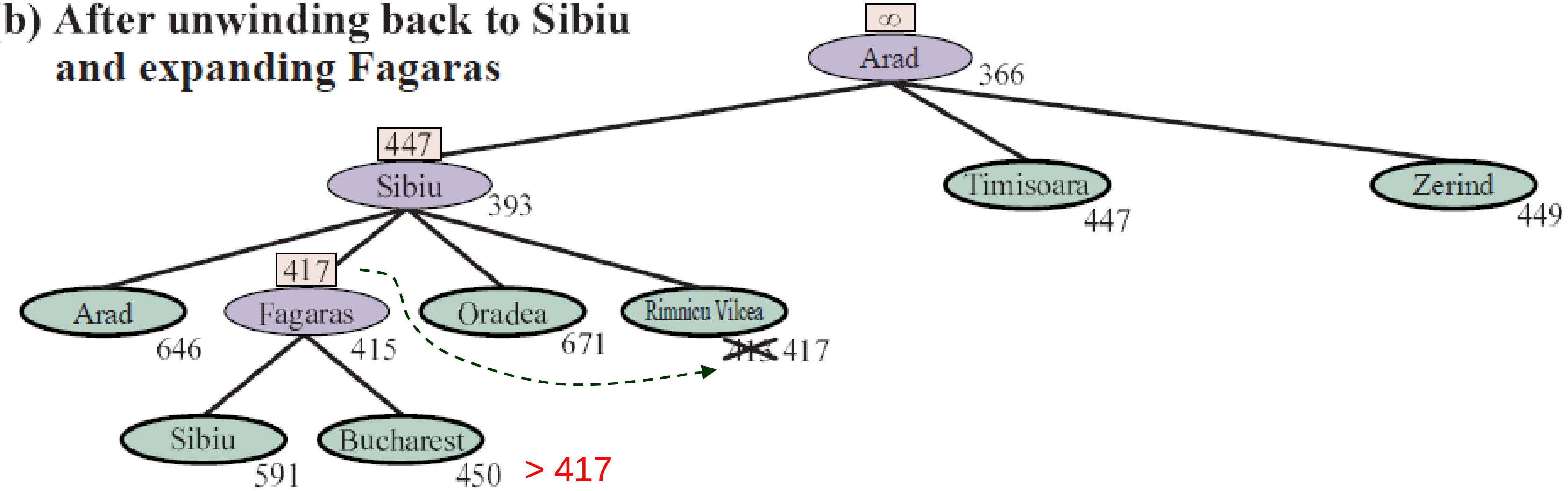


RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

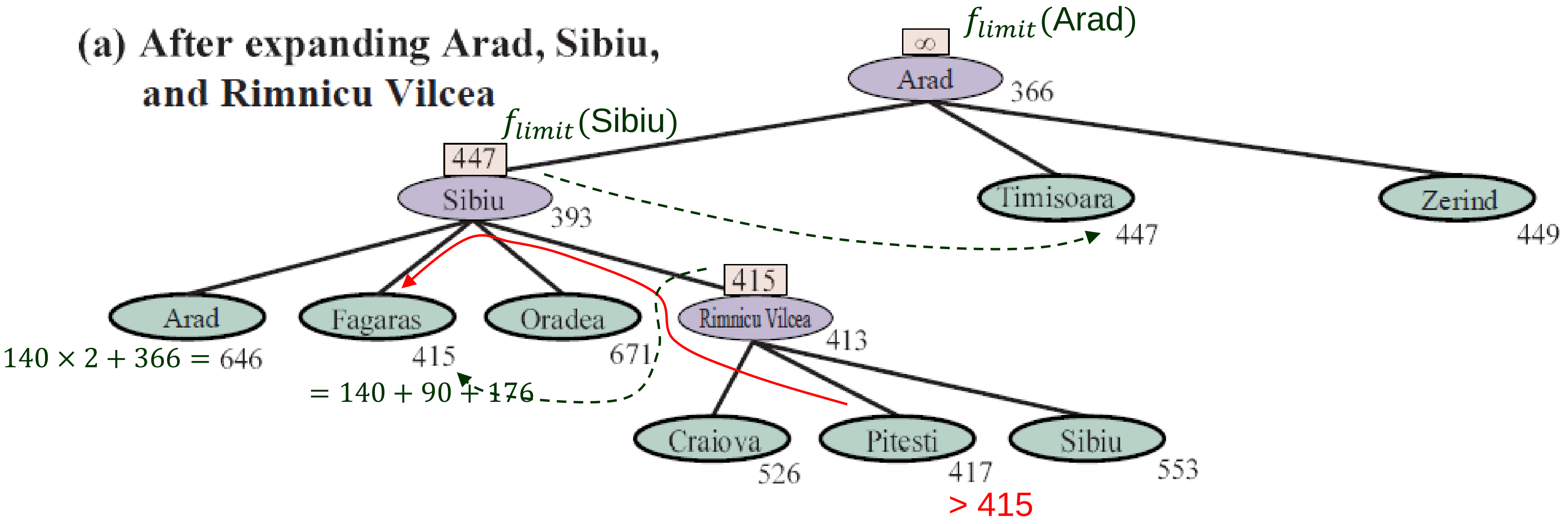


(b) After unwinding back to Sibiu and expanding Fagaras

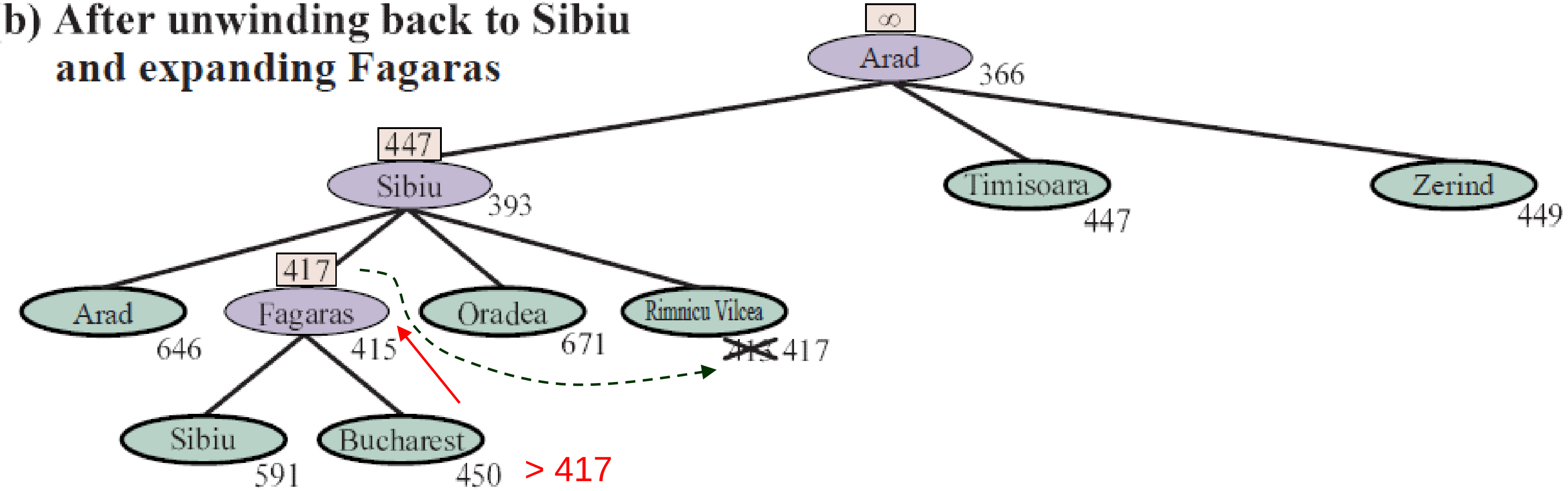


RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

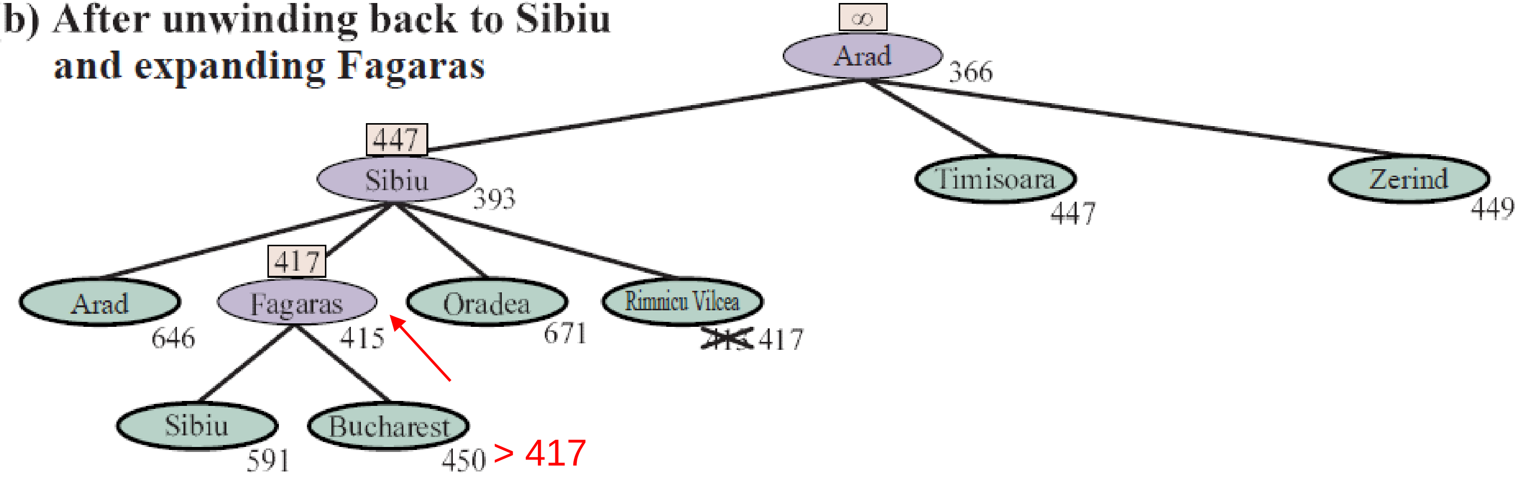


(b) After unwinding back to Sibiu and expanding Fagaras

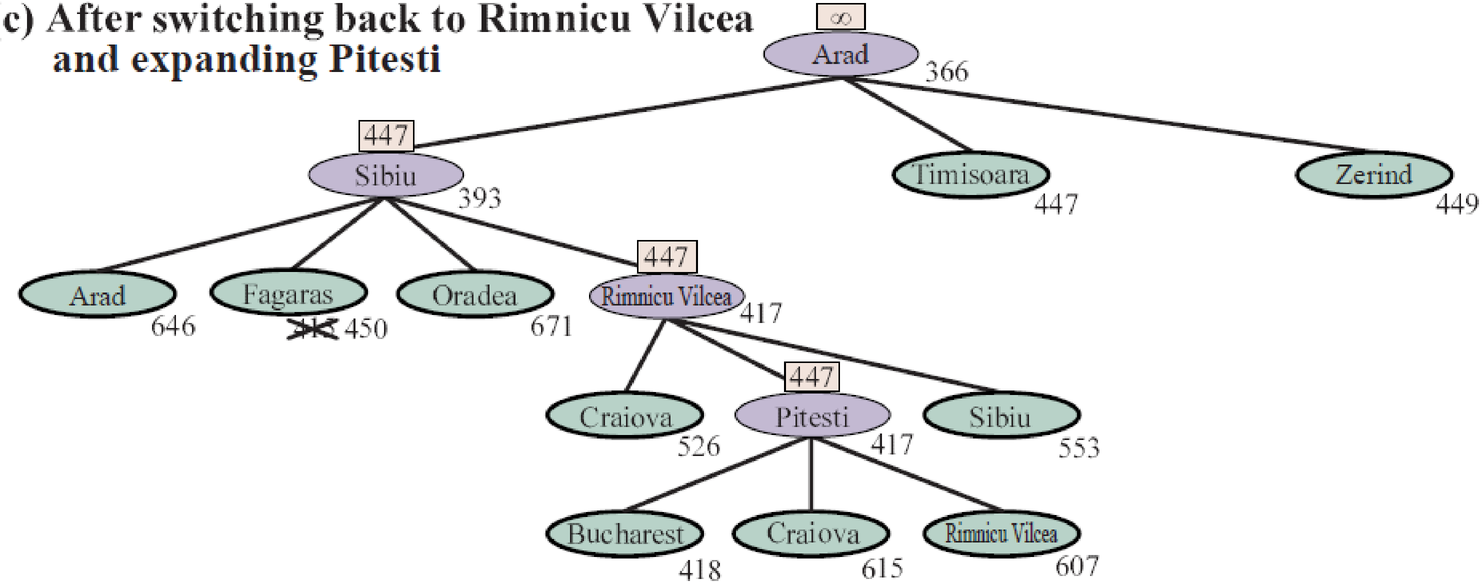


RBFS Example (cont'd)

(b) After unwinding back to Sibiu and expanding Fagaras

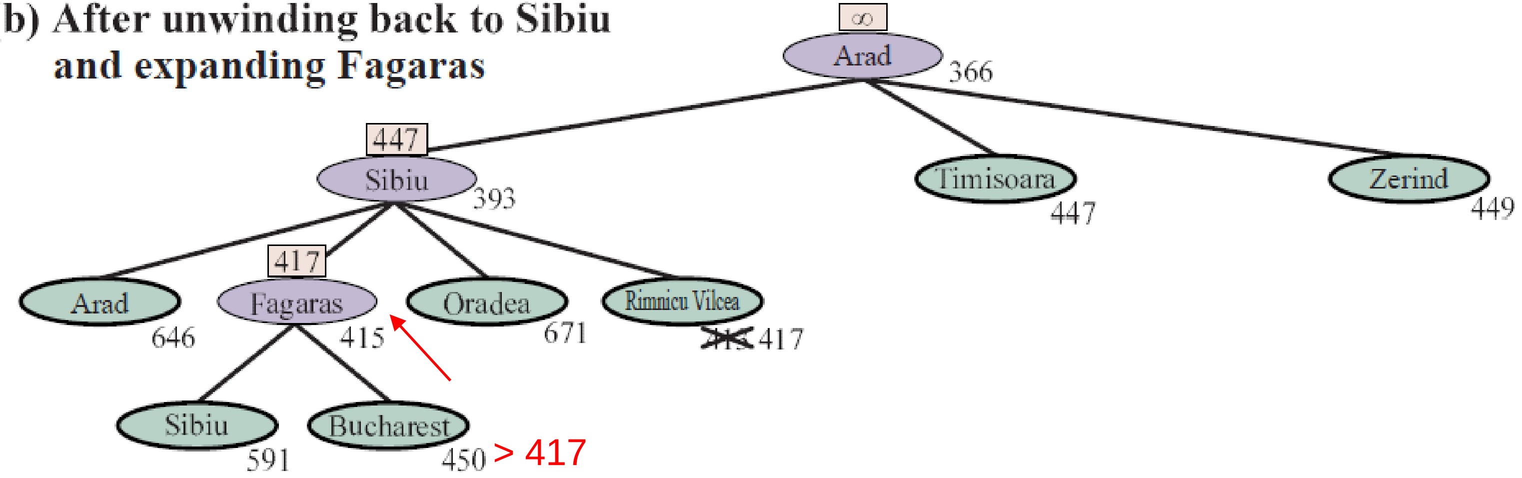


(c) After switching back to Rimnicu Vilcea and expanding Pitesti

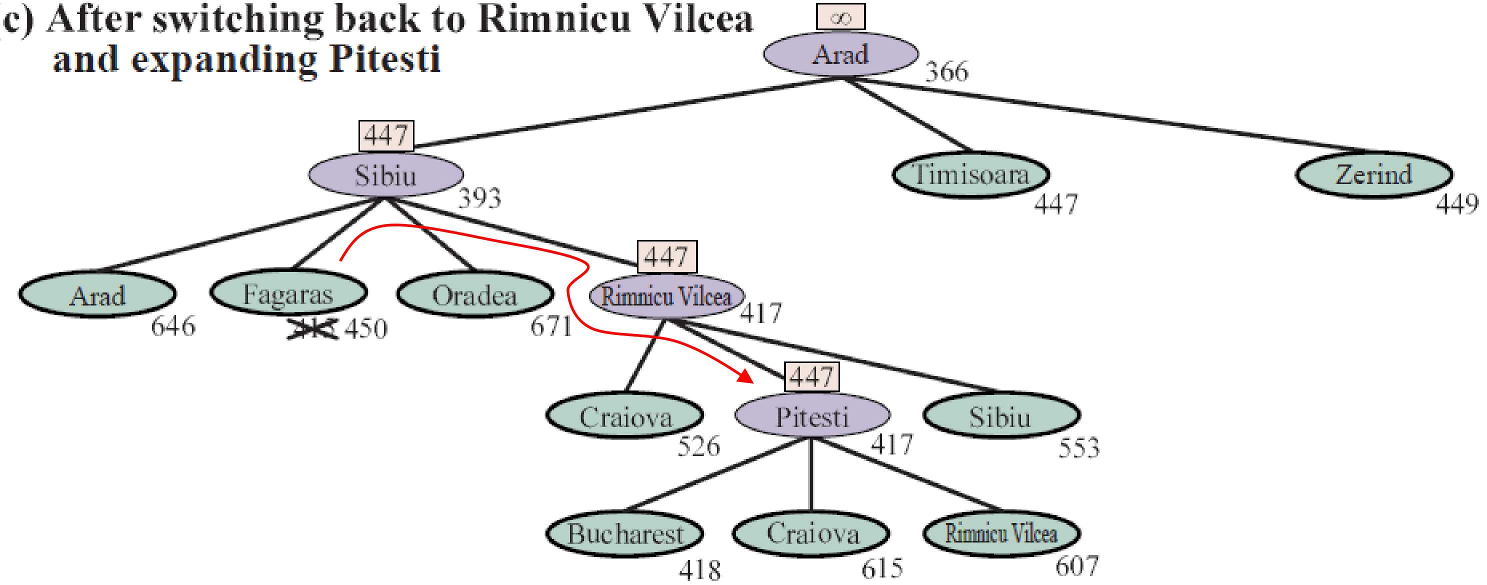


RBFS Example (cont’d)

(b) After unwinding back to Sibiu and expanding Fagaras

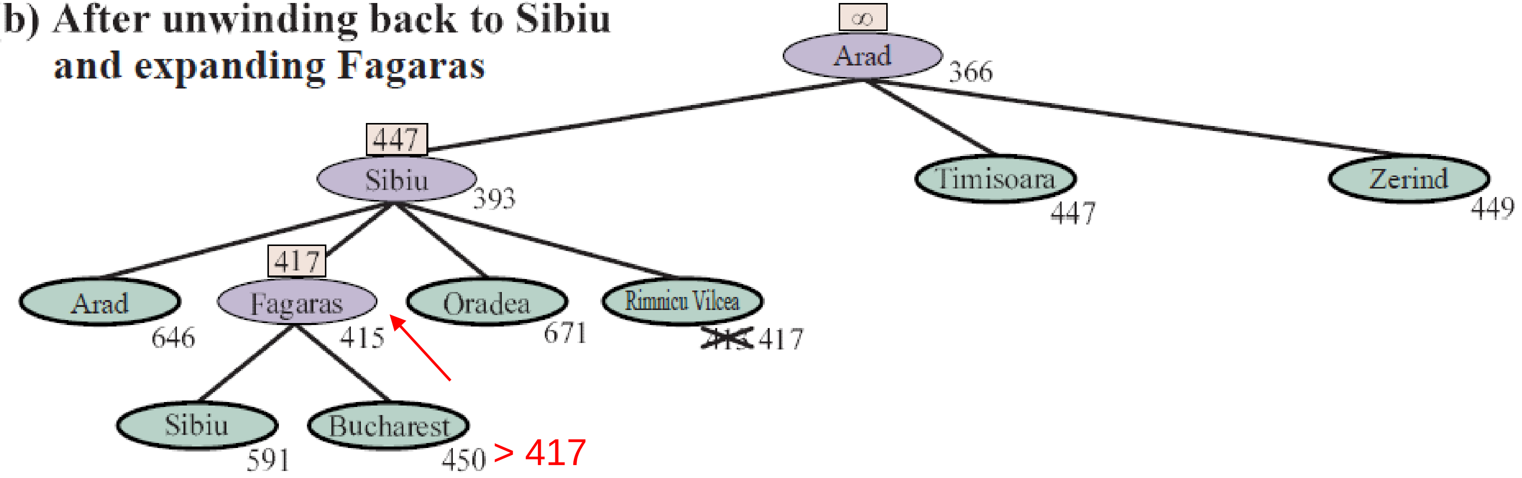


(c) After switching back to Rimnicu Vilcea and expanding Pitesti

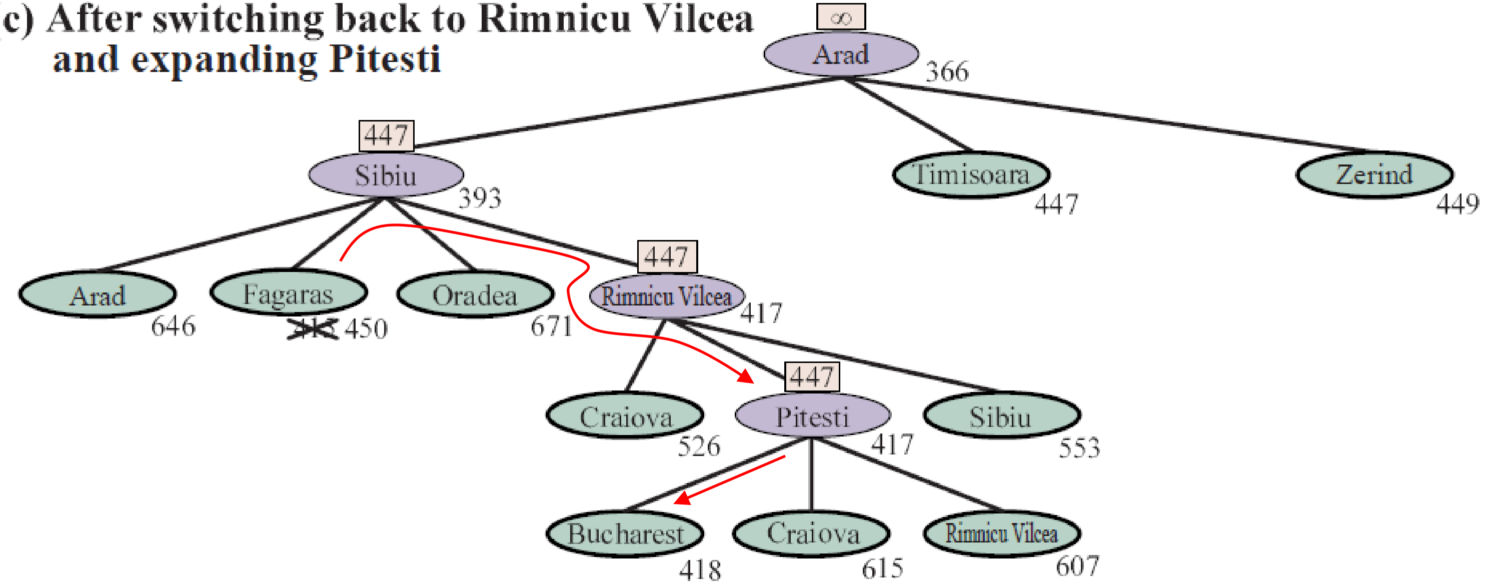


RBFS Example (cont’d)

(b) After unwinding back to Sibiu and expanding Fagaras



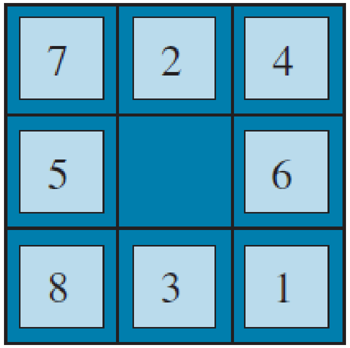
(c) After switching back to Rimnicu Vilcea and expanding Pitesti



RBFS Summary

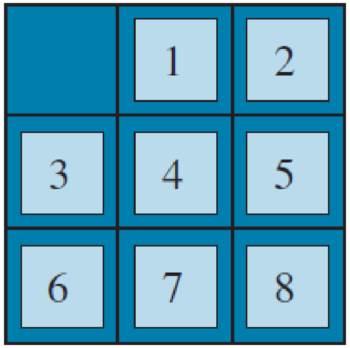
- Optimal with admissible heuristic function $h(n)$.
- Space complexity $O(\underset{\uparrow\ \text{branching factor}}{b}\underset{\uparrow\ \text{depth}}{d})$.
- Time complexity difficult to analyze, depending on
  - accuracy of $h(n)$
  - how often the best path changes
- Slightly more efficient than IDA*.
- Both IDA* and RBFS suffering from using too little memory and may explore the same state multiple times.

III. 8-Puzzle: Large Search Space



7	2	4
5		6
8	3	1

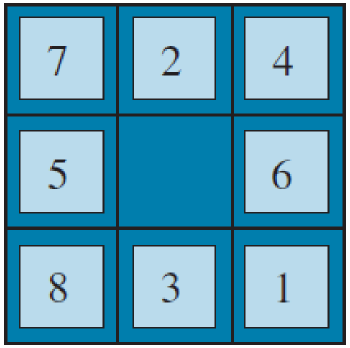
Start State



	1	2
3	4	5
6	7	8

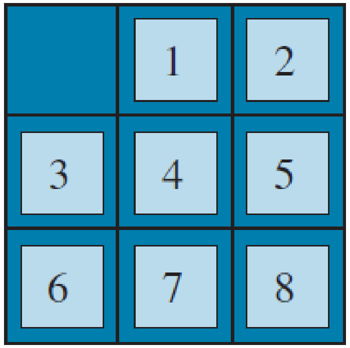
Goal State

III. 8-Puzzle: Large Search Space



7	2	4
5		6
8	3	1

Start State

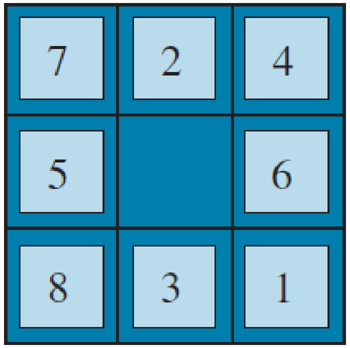


	1	2
3	4	5
6	7	8

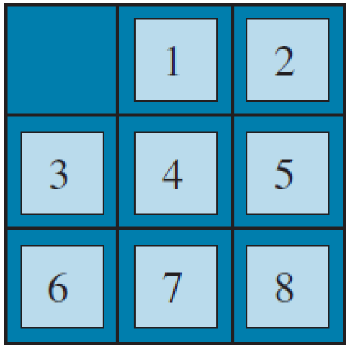
Goal State

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

III. 8-Puzzle: Large Search Space



Start State

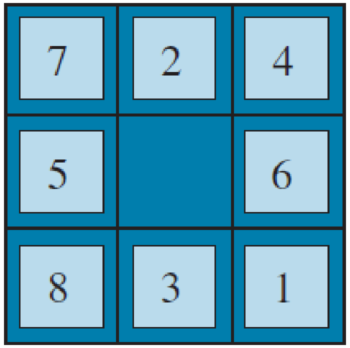


Goal State

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

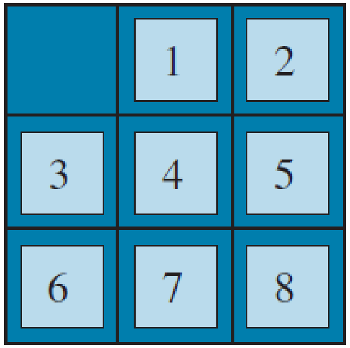
$\frac{16!}{2} > 10^{13}$ reachable states for the 15-puzzle!

Two Heuristics



7	2	4
5		6
8	3	1

Start State

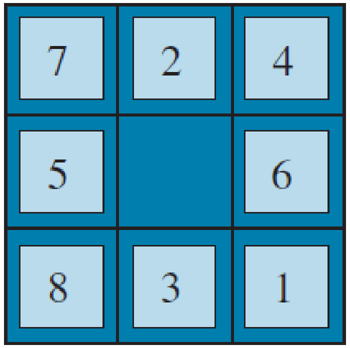


	1	2
3	4	5
6	7	8

Goal State

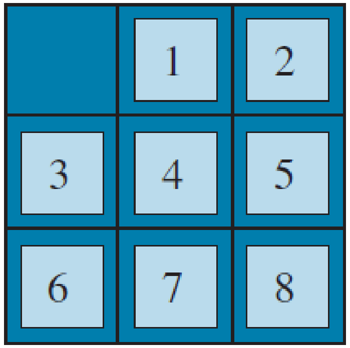
Heuristics are needed for searching the vast state space.

Two Heuristics



7	2	4
5		6
8	3	1

Start State



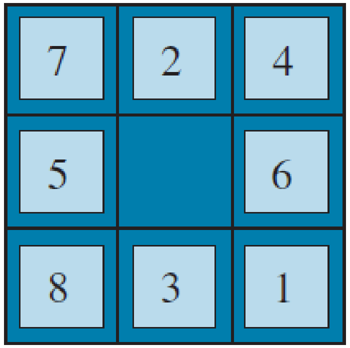
	1	2
3	4	5
6	7	8

Goal State

Heuristics are needed for searching the vast state space.

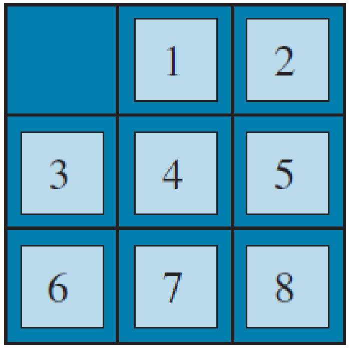
- h_1 = # tiles misplaced

Two Heuristics



7	2	4
5		6
8	3	1

Start State



	1	2
3	4	5
6	7	8

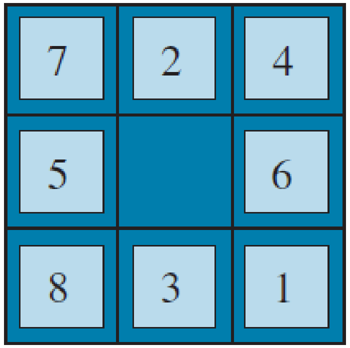
Goal State

Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

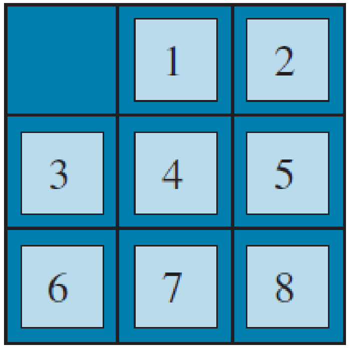
Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8

Two Heuristics



7	2	4
5		6
8	3	1

Start State



	1	2
3	4	5
6	7	8

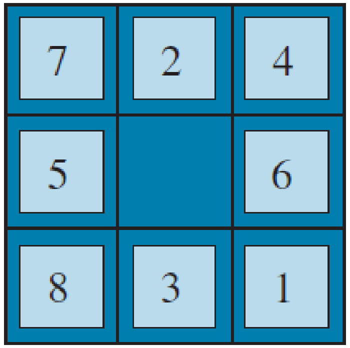
Goal State

Heuristics are needed for searching the vast state space.

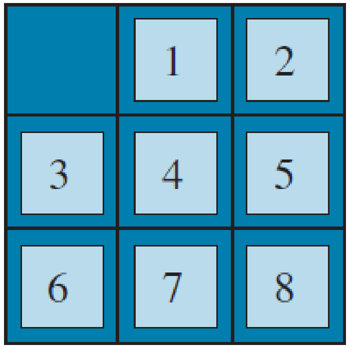
- ♦ h_1 = # tiles misplaced

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

Two Heuristics



Start State



Goal State

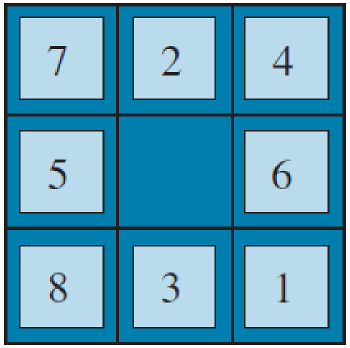
Heuristics are needed for searching the vast state space.

♦ h_1 = # tiles misplaced

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

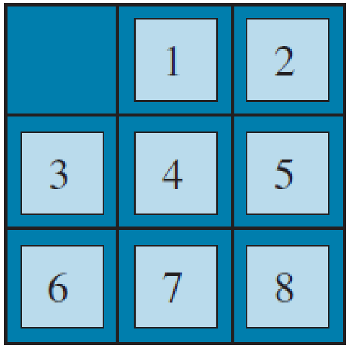
Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8 ⇨ $h_1 = 8$

Two Heuristics



7	2	4
5		6
8	3	1

Start State



	1	2
3	4	5
6	7	8

Goal State

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

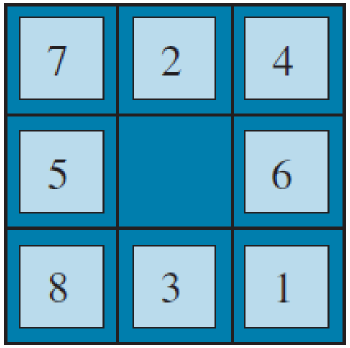
Heuristics are needed for searching the vast state space.

◆ h_1 = # tiles misplaced

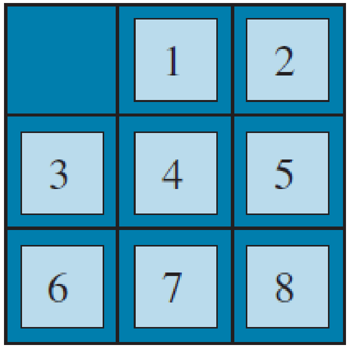
Admissible: any tile out of place will require ≥ 1 move to fix.

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

Two Heuristics



Start State



Goal State

Heuristics are needed for searching the vast state space.

- h_1 = # tiles misplaced

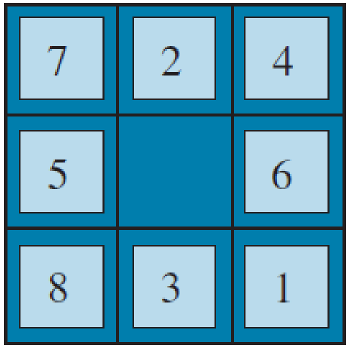
  Admissible: any tile out of place will require $\geq$ 1 move to fix.

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

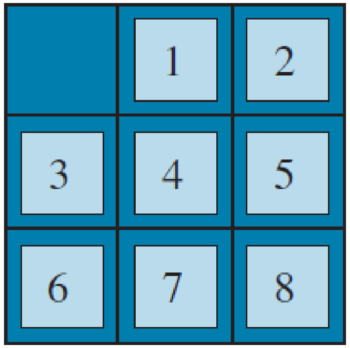
- h_2 = sum of Manhattan distances of the tiles from their goal positions

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

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

⇩

$h_2 = 18$

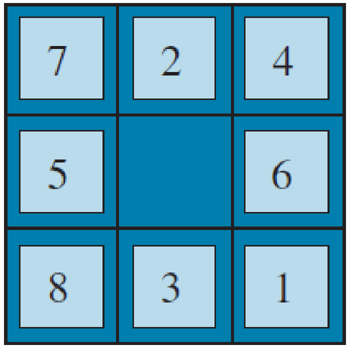
Heuristics are needed for searching the vast state space.

- ♦ h_1 = # tiles misplaced

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

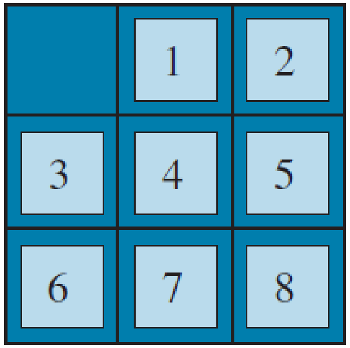
- ♦ h_2 = sum of Manhattan distances of the tiles from their goal positions

Two Heuristics



7	2	4
5		6
8	3	1

Start State



	1	2
3	4	5
6	7	8

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.

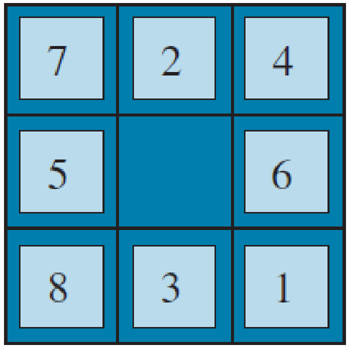
- ♦ h_1 = # tiles misplaced

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

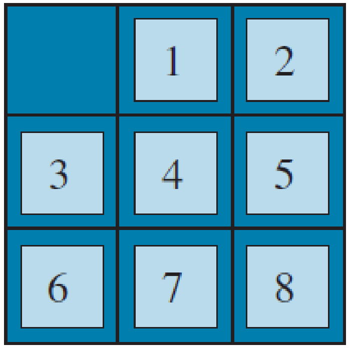
- ♦ h_2 = sum of Manhattan distances of the tiles from their goal positions

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

Two Heuristics



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.

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:

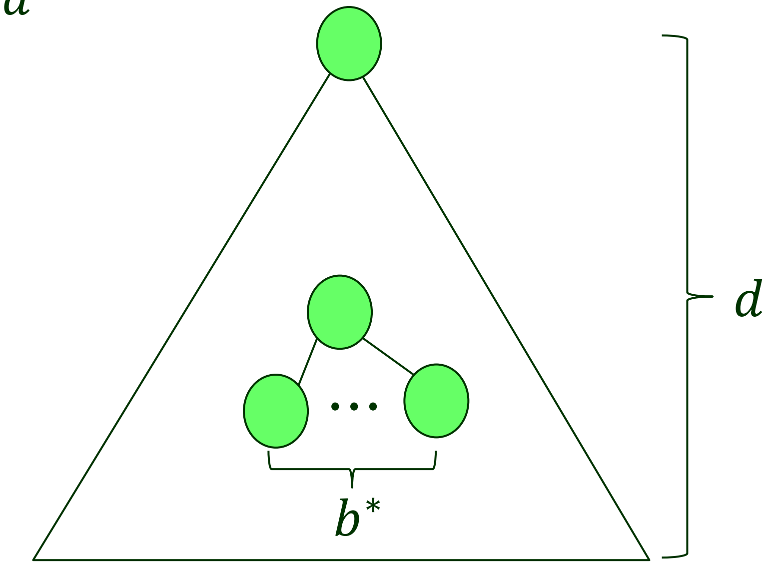
$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

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:

$$N + 1 = 1 + \ b^* + (b^*)^2 + \cdots + (b^*)^d$$



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

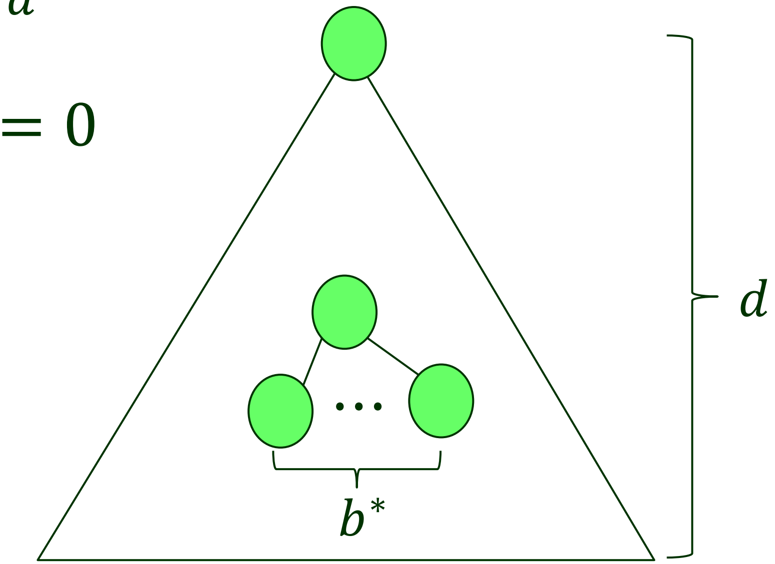
Quality of a heuristic is often measured by the effective branching factor.

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:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

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

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.

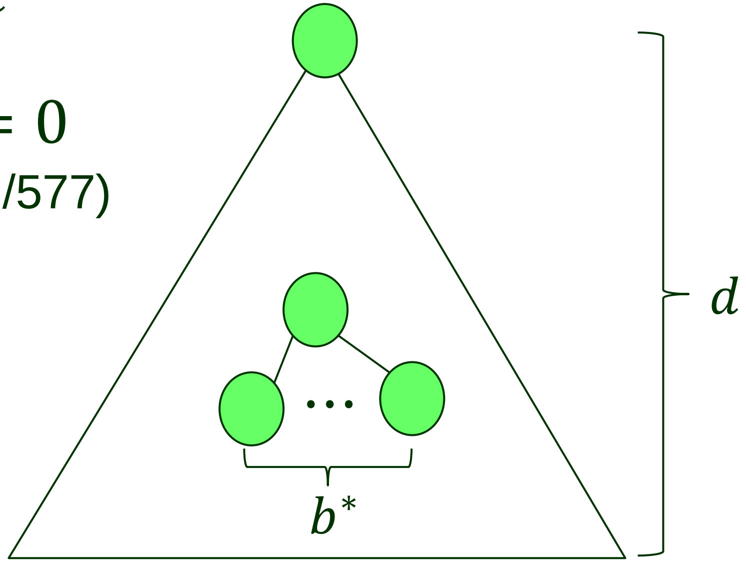
If A* generates N nodes to find a solution at depth d, then its *effective branching factor* b^* is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

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

(polynomial root finding – Com S 477/577)

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

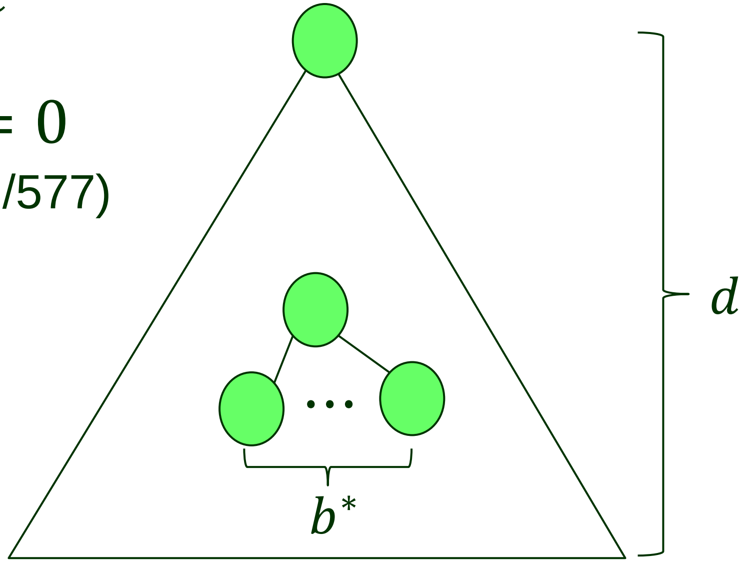
Quality of a heuristic is often measured by the effective branching factor.

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:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

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

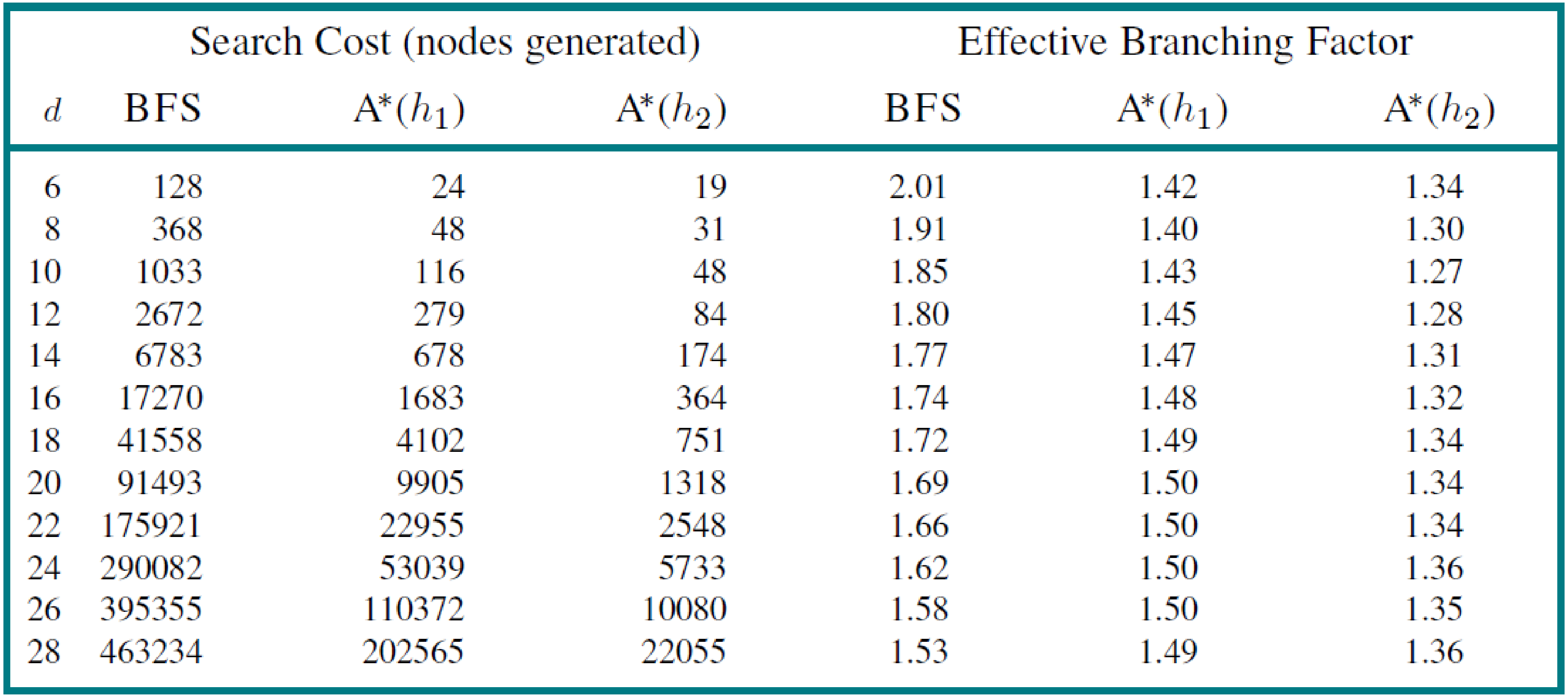
(polynomial root finding – Com S 477/577)



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.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

$$\Updownarrow$$

A* expands every node n with $h(n) < C^* - g(n)$.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

$$\Updownarrow$$

A* expands every node n with $h(n) < C^* - g(n)$.

Suppose that a node n is expanded by A* with h_2, and $h_2(n) < C^* - g(n)$.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \ge h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

⇕

A* expands every node n with $h(n) < C^* - g(n)$.

Suppose that a node n is expanded by A* with h_2, and $h_2(n) < C^* - g(n)$.

⇓

$h\ 1\ n \le\ h\ 2\ n <\ C * - g\ n$

$*\ 2\ 2\ 2\ 1\ 1\ 1\ (n) \le h_2(n) < C^* - g(n)$

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

⇕

A* expands every node n with $h(n) < C^* - g(n)$.

Suppose that a node n is expanded by A* with h_2, and $h_2(n) < C^* - g(n)$.

⇓

$h\ 1\ \ n \leq\ \ h\ 2\ \ n <\ \ C * - g\ n$

$*\ \ 2\ 2\ 2\ 1\ 1\ 1\ (n) \leq h_2(n) < C^* - g(n)$

⇓

The node n will be expanded by A* with h_1.

#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions h_1 and h_2, we say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ at every node n.

If h_2 dominates h_1, A* using h_2 will not expand more nodes than using h_1.

A* expands every node n with $f(n) < C^*$ (optimal cost).

$$\Updownarrow$$

A* expands every node n with $h(n) < C^* - g(n)$.

Suppose that a node n is expanded by A* with h_2, and $h_2(n) < C^* - g(n)$.

$$\Downarrow$$

$$h\,1\;\;n \leq \;\; h\,2\;\;n < \;\; C * \;- g\;n$$

$$* \;\; 2\;2\;2\;1\;1\;1\;(n) \leq h_2(n) < C^* - g(n)$$

$$\Downarrow$$

The node n will be expanded by A* with h_1.

- ♣ h_1 might cause other nodes to be expanded as well.

Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a *relaxed problem*.

with fewer restrictions on the actions

Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a *relaxed problem*.

↑

with fewer restrictions on the actions

Relaxation 1: A tile can move anywhere.

Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a *relaxed problem*.

↑

with fewer restrictions on the actions

Relaxation 1: A tile can move anywhere.

⇓

h_1 would give the length of the shortest solution.

Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a *relaxed problem*.

↑

with fewer restrictions on the actions

Relaxation 1: A tile can move anywhere.

⇓

h_1 would give the length of the shortest solution.

Relaxation 2: A tile can move one square in any direction, even onto an occupied square.

Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a *relaxed problem*.

↑

with fewer restrictions on the actions

Relaxation 1: A tile can move anywhere.

⇓

h_1 would give the length of the shortest solution.

Relaxation 2: A tile can move one square in any direction, even onto an occupied square.

⇓

h_2 would give the length of the shortest solution.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

⇩

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

⇩

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

- Such heuristic is an exact cost for the relaxed problem.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

⇩

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

- Such heuristic is an exact cost for the relaxed problem.

⇩

It must satisfy the triangle inequality.

Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

⇩

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

- Such heuristic is an exact cost for the relaxed problem.

⇩

It must satisfy the triangle inequality.

⇩

The heuristic is consistent.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

> A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇩ Relaxation by removing one or two conditions

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⇒ h_2 (Manhattan distance)

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⇒ h_2 (Manhattan distance)

(b) A tile can move from square X to square Y if Y is blank.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

$\Downarrow$ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. $\Rightarrow$ h_2 (Manhattan distance)

(b) A tile can move from square X to square Y if Y is blank.

(c) A tile can move from square X to square Y.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⇒ h_2 (Manhattan distance)

(b) A tile can move from square X to square Y if Y is blank.

(c) A tile can move from square X to square Y. ⇒ h_1 (misplaced tiles)

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⟹ h_2 (Manhattan distance)

↓

(b) A tile can move from square X to square Y if Y is blank.

(c) A tile can move from square X to square Y. ⟹ h_1 (misplaced tiles) → Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇓ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⇒ h_2 (Manhattan distance) ↓

(b) A tile can move from square X to square Y if Y is blank.

(c) A tile can move from square X to square Y. ⇒ h_1 (misplaced tiles) → Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.

- ♦ The relaxed problems should be solved without search. Otherwise, evaluation of the corresponding heuristic will be expensive.

Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square X to square Y if X is adjacent to Y and Y is blank.

⇩ Relaxation by removing one or two conditions

(a) A tile can move from square X to square Y if X is adjacent to Y. ⟹ h_2 (Manhattan distance)

(b) A tile can move from square X to square Y if Y is blank.

(c) A tile can move from square X to square Y. ⟹ h_1 (misplaced tiles) → Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.

- The relaxed problems should be solved without search.

  Otherwise, evaluation of the corresponding heuristic will be expensive.

- Program ABSOLVER generates heuristics automatically from problem definitions, including the best one for the 8-puzzle and the first one for the Rubik's Cube puzzle.

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

- h is *admissible* because $h_1, h_2, \cdots, h_k$ are.

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

- h is *admissible* because $h_1, h_2, \cdots, h_k$ are.
- h *dominates* $h_1, h_2, \cdots, h_k$.

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

- ♦ h is *admissible* because $h_1, h_2, \cdots, h_k$ are.
- ♦ h *dominates* $h_1, h_2, \cdots, h_k$.
- ♣ h takes longer to compute – at least $O(k)$.
  May randomly select one heuristic at each evaluation.

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

- ♦ h is *admissible* because $h_1, h_2, \cdots, h_k$ are.
- ♦ h *dominates* $h_1, h_2, \cdots, h_k$.
- ♣ h takes longer to compute – at least $O(k)$.
  May randomly select one heuristic at each evaluation.