Artificial Intelligence

Informed Search

Informed (Heuristic) Search and Admissible Heuristics

Lecture 5

CS 444 – Spring 2019

Dr. Kevin Molloy

Department of Computer Science

James Madison University

Outline for Today

- Searching, is there a better way?

- Setup for Project 1

Summary of Uninformed Search Algorithms

Criterion	BFS	DFS	DLS	IDS
Complete?	Yes	No	Yes if dl ≥ d	Yes
Time	b^{d+1}	b^m	b^{dl}	b^d
Space	b^{d+1}	bm	bd_l	bd
Optimal?	Yes*	No	No	Yes*

Method for Uninformed Searches

Insight: All covered graph-search algorithms follow similar template:

- “Maintain" a set of explored vertices S and a set of unexplored vertices V - S
- “Grow" S by exploring edges with exactly one endpoint in S and the other in V - S
- What do we actually store in the fringe?

Implication: similar template ! reusable code

Data structure ***F*** for the fringe: order vertices are extracted from V - S distinguishes

- search algorithms from one another
- DFS: Take edge from vertex discovered most recently (F is a stack)
- BFS: Take edge from vertex discovered least recently (F is a queue)

Informed Search Algorithms

Find a **least-cost/shortest** path from initial vertex to goal vertex. These exploit cost/weights in the state-space graph.

Informed graph search algorithms:

- Dijkstra's [1959]
- Uniform-cost Search (variant of Dijkstra's)
- Best-First Search [Pearl 1984]
- A* Search [Hart, Nilsson, Raphael, 1968]
- B*Search [Berliner 1979]
- D* Search [Stenz 1994]

We are going to skip cover cases with negative weights.

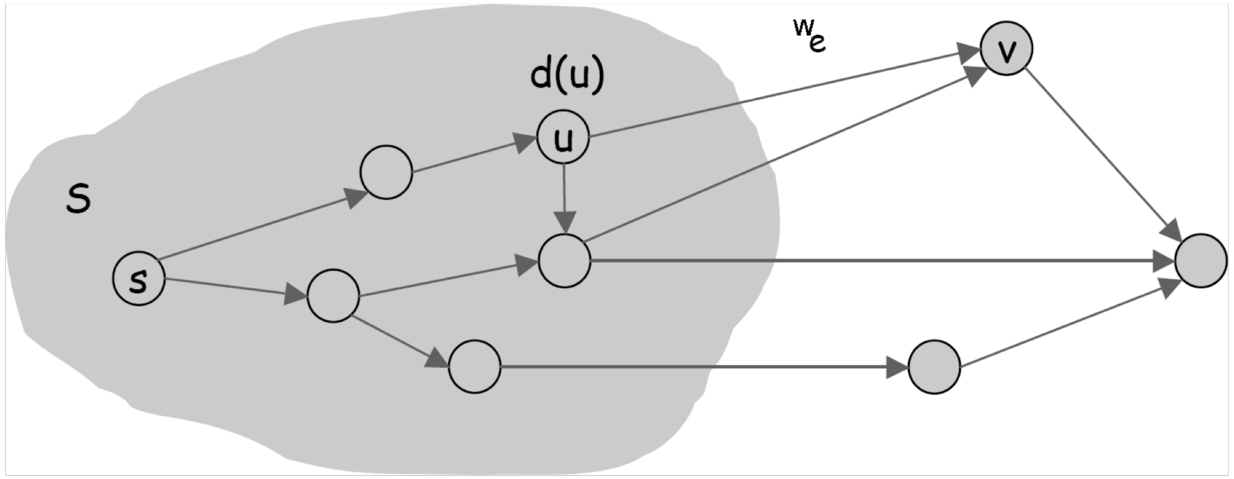
Finding Shortest Paths in Weighted Graphs

The **weight of a path** *p* = (v_1, v_2, … v_k) is the sum of the weights of the corresponding edges: $w(p) = \sum_{i=1}^{k} w(v_{i-1},v_i)$.

The **shortest path weight** from a vertex u to a vertex v is:

$$\delta(u,v) = \begin{cases} \min\{w(p): p = (u,\dots,v)\} & \textit{if p exists} \\ \infty & \textit{otherwise} \end{cases}$$

- A **shortest path** from *u* to *v* is any path *p* with weight $\delta(u,v)$
- The **tree of shortest paths** is a spanning tree of *G* = (*V, E*), where the path from its root, the source vertex *s*, to any vertex *u* ∈ *V* is the shortest path *s*↝ *u* in *G*.



- Tree grows from *S* to *V – S*
- Start vertex first to be extract from V-S and added to *S*
- As S grows (V – S shrinks), tree grows
- Tree grows in iterations, one vertex extracted from V - S at a time

Essence of all Informed Search Algorithms

All you need to remember about informed search algorithms

- Associate a(n attachment) cost d[v] with each vertex v
- *F* becomes a priority queue: *F* keeps frontier vertices, prioritized by d[v]
- Until F is empty, one vertex extracted from *F* at a time
    - Can terminate earlier? When? How does it relate to goal?
- v extracted from F @ some iteration is one with lowest cost among all those in F
    - ... so, vertices extracted from F in order of their costs
- When *v* extracted from *F*:
    - v has been “removed” from V - S and “added” to S
    - get to reach/see v's neighbors and possibly update their costs

Essence of all Informed Search Algorithms

The rest are details, such as:

- What should d[v] be? There are options...
- backward cost (cost of $s \rightsquigarrow v$)
- forward cost (estimate of cost of $v \rightsquigarrow g$)
- back+forward cost (estimate of $s \rightsquigarrow g$ through v)

Which do I choose? This is how to you end up with different search algorithms

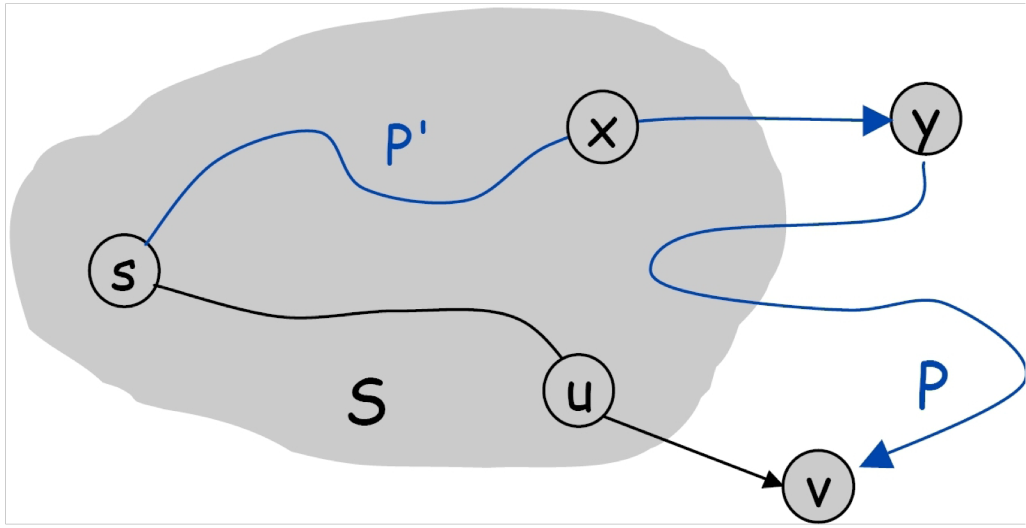
Dijkstra’s Shortest Path Algorithm

Dijkstra extracts vertices from the frontier (adds to S) in order of their costs

Claim: When a vertex *v* is extracted from the fringe F (i.e., added to S), the shortest path from *s* to *v* has been found.

Proof: by induction on |S| (base case |S| =1 is trivial). Assume invariant holds for |S| = k > 1

- Let *v* be vertex about to be extracted from the fringe (added to S), so, has lowest backwards cost
- Last time d[v] updated when parent *u* extracted from fringe
- When d[v] is the lowest in the fringe, should we extract v or wait?
- Could d[v] get lower later through some other vertex y in the fringe?



$W(p) \geq w(P') + w(x,y)$	Nonnegative weights
$\geq d[x] + w(x,y)$	Inductive hypothesis
$\geq d[y]$	Definition of d[y]
$\geq d[v]$	Dijkstra chose v over y

JMU JAMES MADISON UNIVERSITY.

Some Quotes from Dijkstra

Edsger Dijkstra 1930 - 2002

The question of whether computers can think is like the questions of whether submarines can swim.

JMU JAMES MADISON UNIVERSITY.

Dijkstra’s Shortest in Pseudocode

Fringe F is a priority queue/min-heap

Arrays: d stores attachments (backwards cost), π[v] stores parent

S only shown for clarity (is not required)

Dijkstra(G, s, w)

```
1: F ← s, s ← ∅
2: d[v] ← ∞ for all v ∈ V
3: d[s] ← 0
4: while F ≠ ∅ do
5:    u ← Extract-Min(F)
6:    S ← S ∪ {u}
7:    for each v ∈ Adj(u) do
8:       F ← F ∪ {v}
9:       Relax (u,v,w)
```

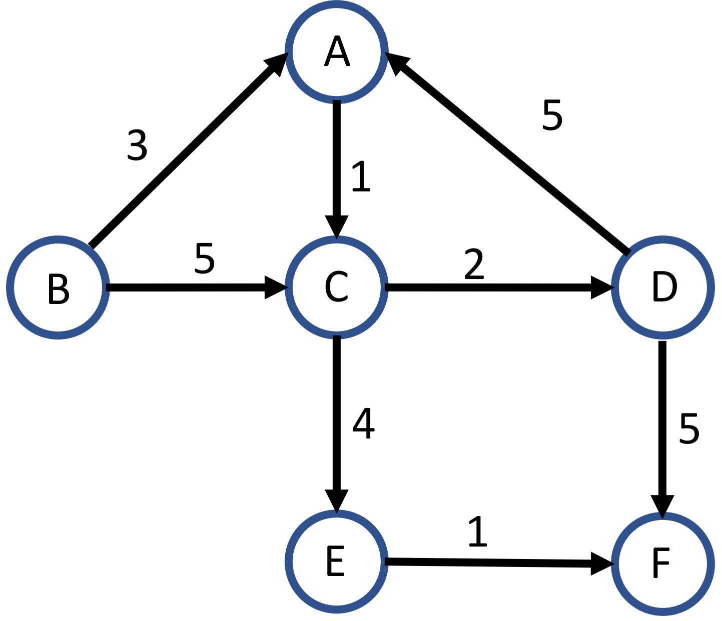
Relax(u, v, w)

```
   If d[v] > d[u] + w(u,v) then
1:    d[v] ← d[u] + w(u,v)
2:    π[v] ← u
3:
```

JAMES MADISON UNIVERSITY

Dijkstra's Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```



u = B

	Init			P 1			P 2			P 3			P 4			P 5			P 6	
Vert	d	π		d	π		d	π		d	π		d	π		d	π		d	π
A	∞																			
B	0	-																		
C	∞																			
D	∞																			
E	∞																			
F	∞																			

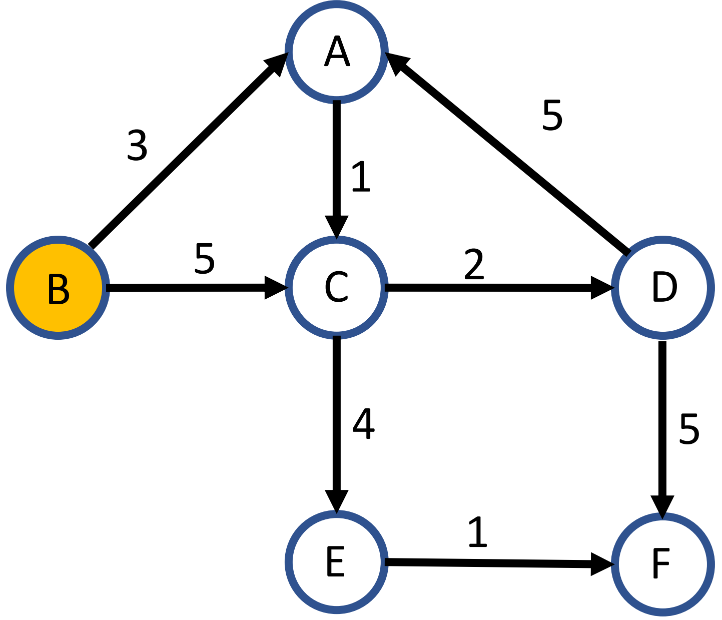
Dijkstra’s Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```

s = B

F = {A(3), C(5)}

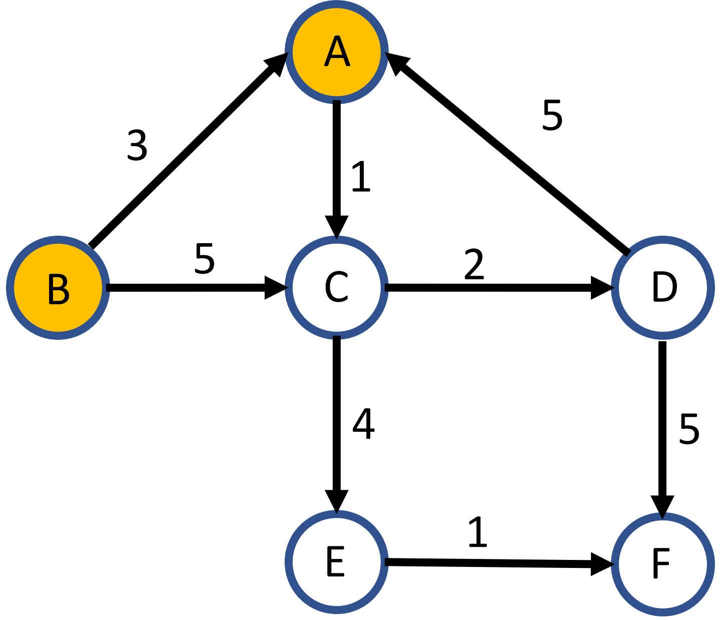
u = B



	Init		P 1		P 2		P 3		P 4		P 5		P 6	
Vert	d	π	d	π	d	π	d	π	d	π	d	π	d	π
A	∞		3	B										
B	0	-	0	-										
C	∞		5	B										
D	∞		∞											
E	∞		∞											
F	∞		∞											

Dijkstra’s Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```



F = {C(4)}

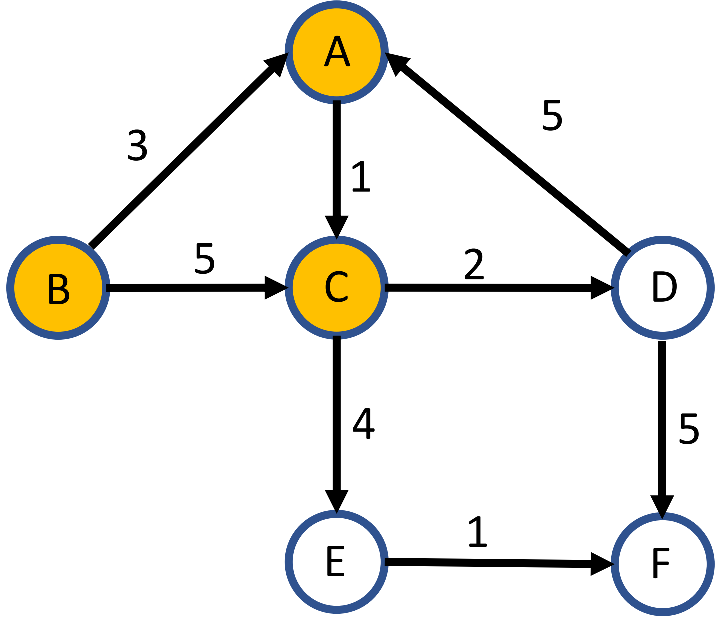
u = B   *u* = A

	Init		P 1		P 2		P 3		P 4		P 5		P 6	
Vert	d	π	d	π	d	π	d	π	d	π	d	π	d	π
A	∞		3	B	3	B								
B	0	-	0	-	0	-								
C	∞		5	C	4	A								
D	∞		∞		∞									
E	∞		∞		∞									
F	∞		∞		∞									

JAMES MADISON UNIVERSITY

Dijkstra’s Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```



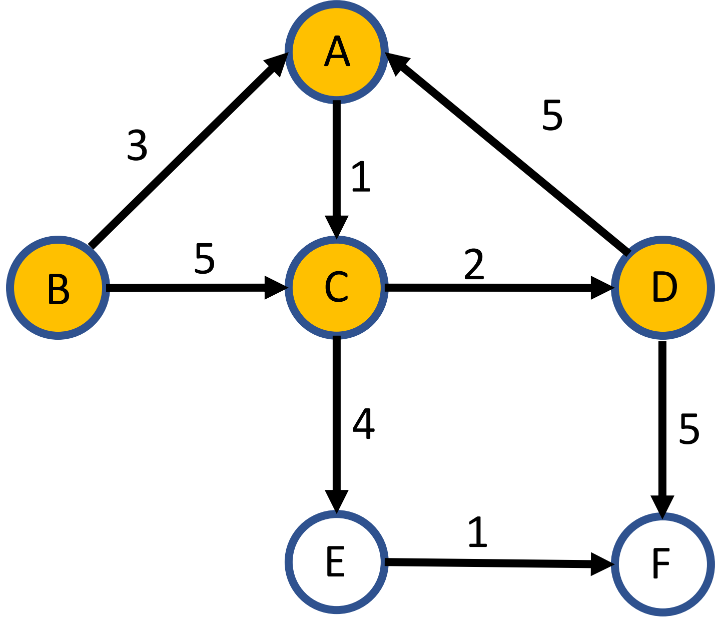
F = {6(D), 8(E)}

u = B   u = A   u = C

	Init			P 1			P 2			P 3			P 4			P 5			P 6	
Vert	d	π		d	π		d	π		d	π		d	π		d	π		d	π
A	∞			3	B		3	B		3	B									
B	0	-		0	-		0	-		0	-									
C	∞			5	C		4	A		4	A									
D	∞			∞			∞			6	C									
E	∞			∞			∞			8	E									
F	∞			∞			∞			∞										

Dijkstra's Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```

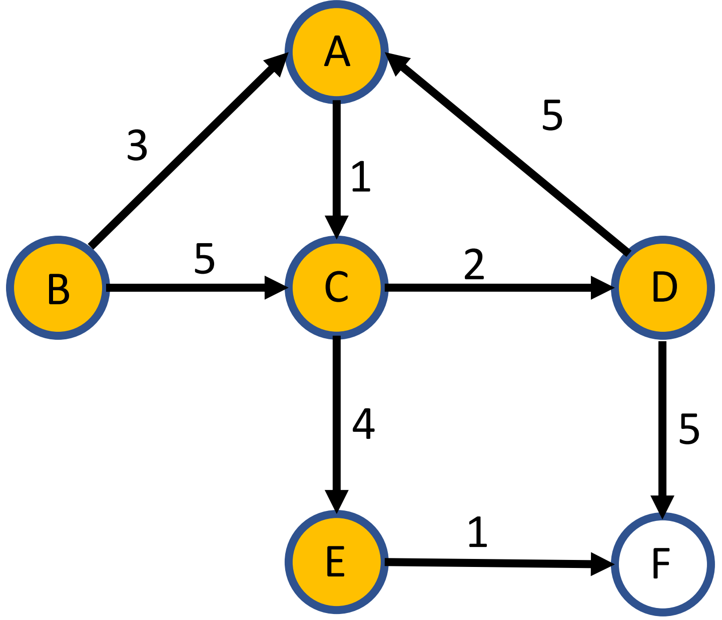


F = {8(E),11(F)}

				u = B			u = A			u = C			u = D						
	Init			**P 1**			**P 2**			**P 3**			**P 4**			**P 5**			**P 6**
Vert	d	π		d	π		d	π		d	π		d	π		d	π		d	π
A	∞			3	B		3	B		3	B		3	B						
B	0	-		0	-		0	-		0	-		0	-						
C	∞			5	C		4	A		4	A		4	A						
D	∞			∞			∞			6	C		6	C						
E	∞			∞			∞			8	E		8	E						
F	∞			∞			∞			∞			11	F						

Dijkstra’s Shortest in Pseudocode

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```



F = {9(F)}

	Init		P 1		P 2		P 3		P 4		P 5		P 6	
			u = B		*u* = A		*u* = C		*u* = D		*u* = E			
Vert	d	π	d	π	d	π	d	π	d	π	d	π	d	π
A	∞		3	B	3	B	3	B	3	B	3	B		
B	0	-	0	-	0	-	0	-	0	-	0	-		
C	∞		5	C	4	A	4	A	4	A	4	A		
D	∞		∞		∞		6	C	6	C	6	C		
E	∞		∞		∞		8	E	8	E	8	E		
F	∞		∞		∞		∞		11	F	9	F		

Dijkstra’s Shortest in Pseudocode

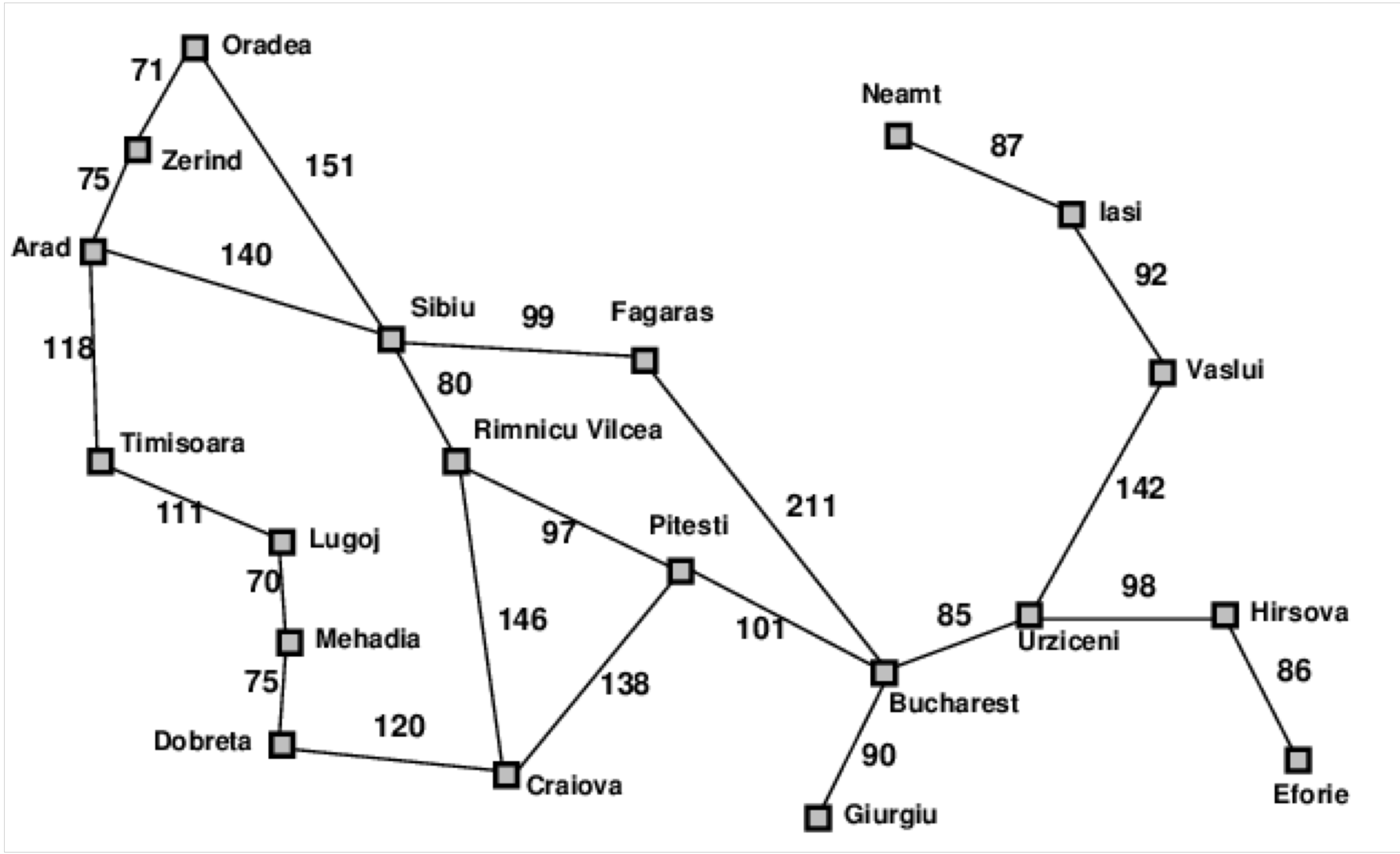
$$\begin{tikzcd}
 & A \arrow[d, "1"] & \\
B \arrow[ur, "3"] \arrow[r, "5"] & C \arrow[r, "2"] \arrow[d, "4"] & D \arrow[ul, "5"'] \arrow[d, "5"] \\
 & E \arrow[r, "1"] & F
\end{tikzcd}$$

```
F ← s, s ← ∅
d[v] ← ∞ for all v ∈ V
d[s] ← 0
while F ≠ ∅ do
    u ← Extract-Min(F)
    S ← S ∪ {u}
    for each v ∈ Adj(u) do
        F ← F ∪ {v}
        Relax (u,v,w)
```

F = {}

			u = B		u = A		u = C		u = D		u = E		u = F	
	Init		P 1		P 2		P 3		P 4		P 5		P 6	
Vert	d	π	d	π	d	π	d	π	d	π	d	π	d	π
A	∞		3	B	3	B	3	B	3	B	3	B	3	B
B	0	-	0	-	0	-	0	-	0	-	0	-	0	-
C	∞		5	C	4	A	4	A	4	A	4	A	4	A
D	∞		∞		∞		6	C	6	C	6	C	6	C
E	∞		∞		∞		8	E	8	E	8	E	8	E
F	∞		∞		∞		∞		11	F	9	E	9	E

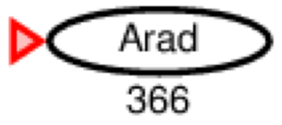
Greedy Best-first Search in Action



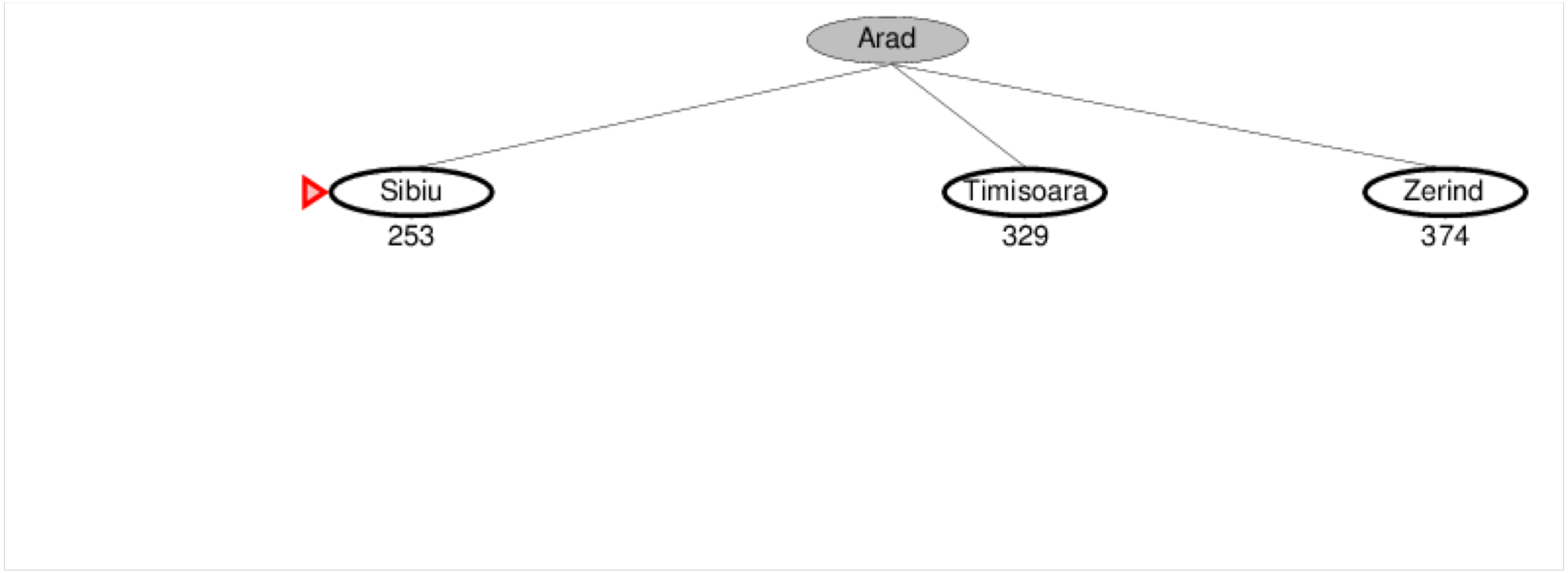
Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

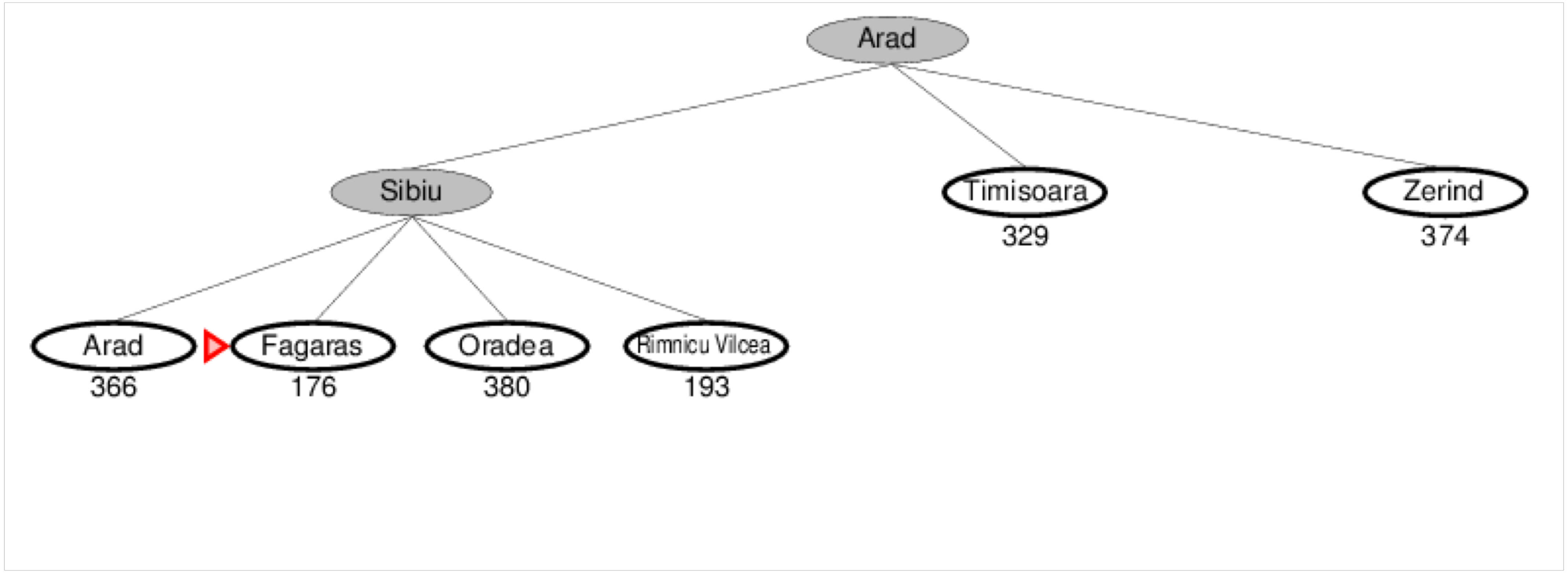
Greedy Best-first Search in Action



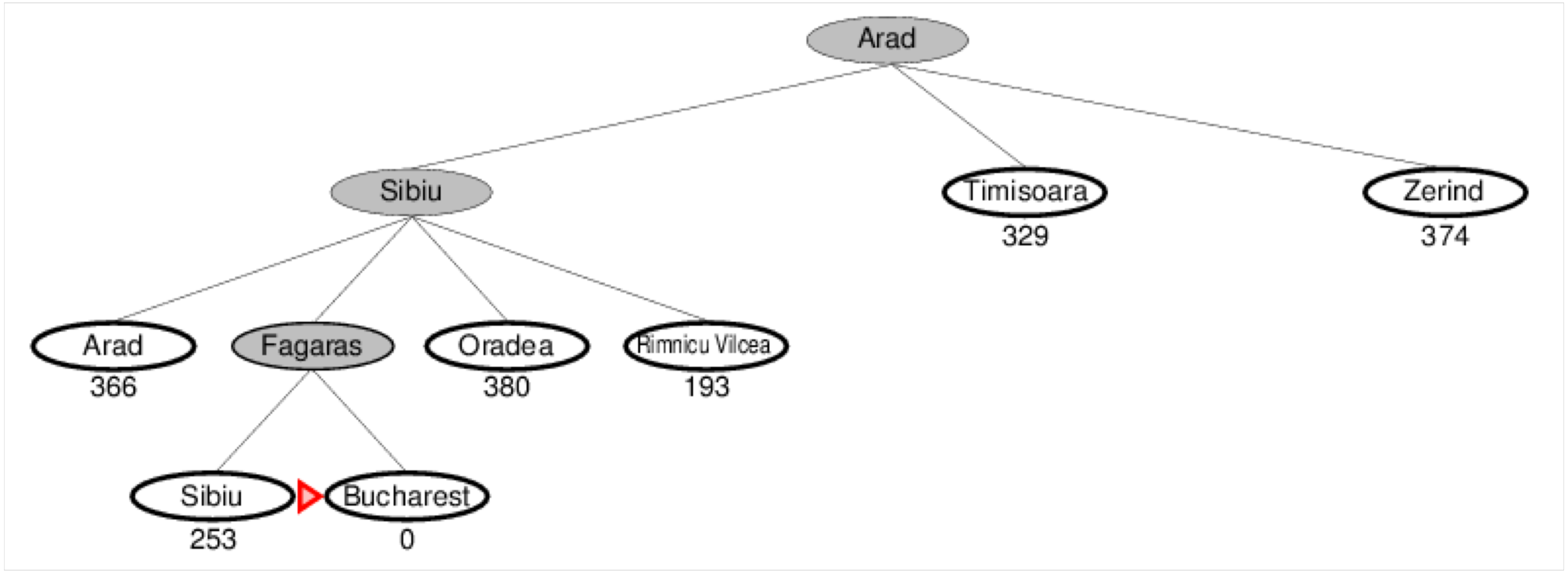
Greedy Best-first Search in Action



Greedy Best-first Search in Action



Greedy Best-first Search in Action



Optimal? No, since we can get to Bucharest Rimnicu Vilcea and Pitesti.

Summary of Greedy Best-first Search

Complete?:	Complete in finite space with repeated state checking
Time?	$O(b^m)$, but a good heuristic can give dramatic improvement
Space?	$O(b^m)$ – keeps all nodes in memory
Optimal?	No

A* Search

Idea: avoid expanding paths that are already expensive.

Evaluation function: f(v) = g(v) + h(v):

Combines Dijkstra's/uniform cost with greedy best-first search

- g(v) = (actual) cost to reach v from s
- h(v) = estimated lowest cost from v to goal
- f(v) = estimated lowest cost from s through v to goal

Same implementation as before, but prioritize vertices in the min-heap by f[v].

A* is both complete and optimal provided h satisfies certain conditions:

- For searching in a tree: admissible/optimistic
- For searching in a graph: consistent (which implies admissibility)

Dijkstra’s Shortest Path Algorithm

What do we want from f[v]?

- Not to overestimate cost of path from source to goal that goes through v

Since g[v] is actual cost from s to v, this “do not overestimate” criterion is for the forward cost heuristic, h[v]

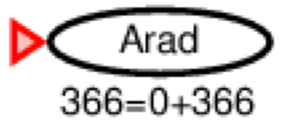
A* searches uses an admissible/optimistic heuristic

i.e., $h(v) \leq h^*(v)$ where $h^*(v)$ is the true cost from v

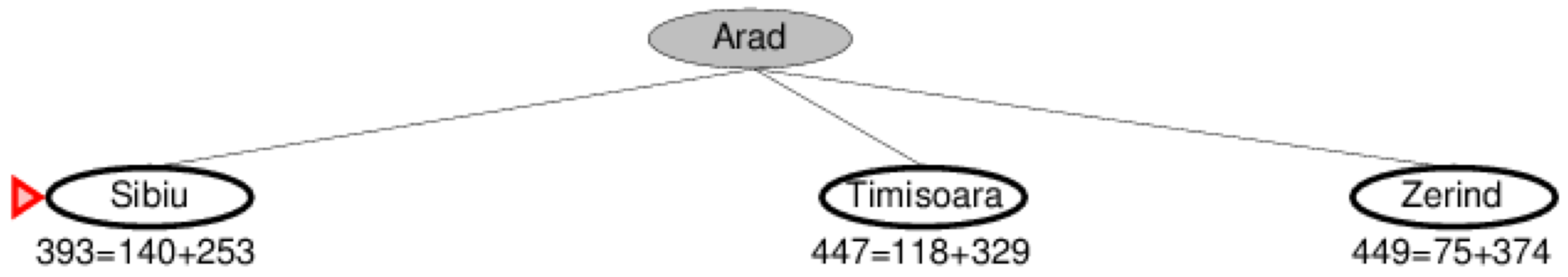
(Also require $h(v) \geq 0$, so $h(G) = 0$ for any goal G).

Example of an admissible heuristic: $h_{sld}(v)$ never overestimates the actual road distance.

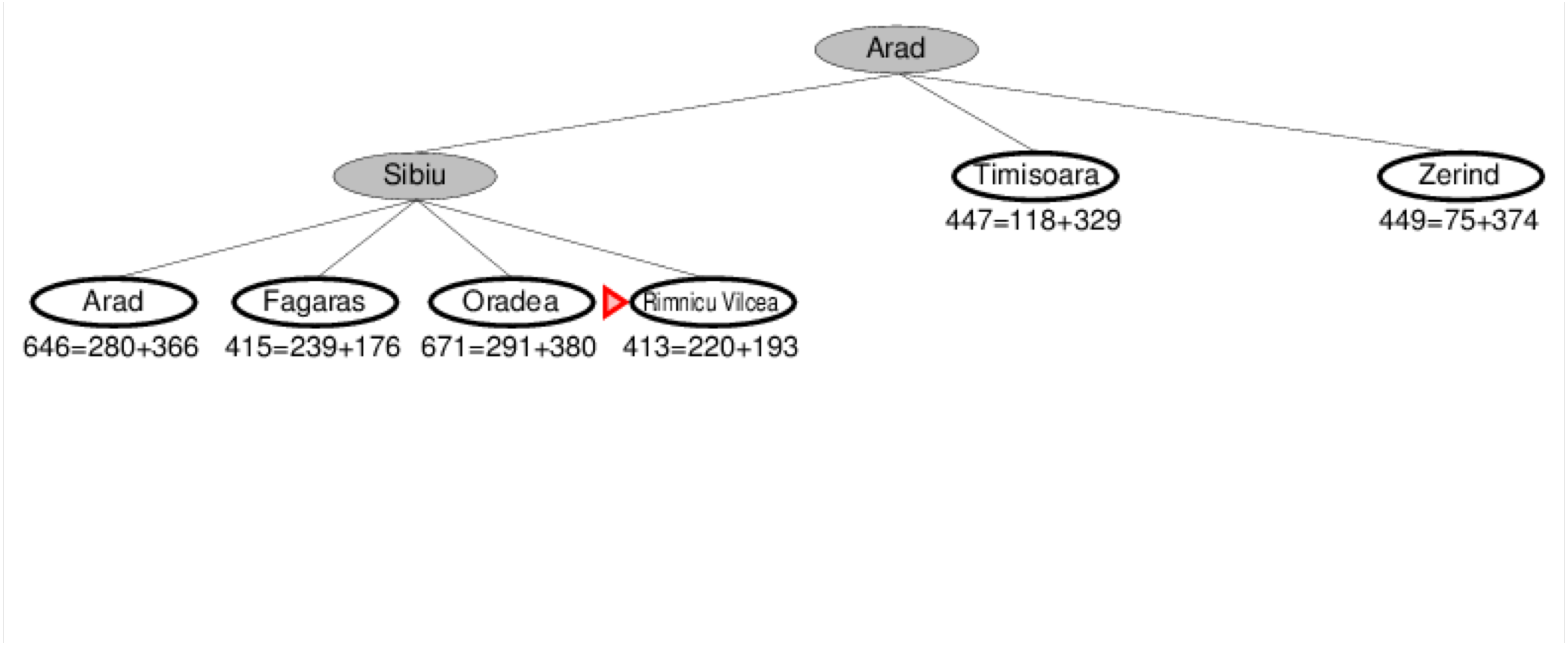
A* Search in Action



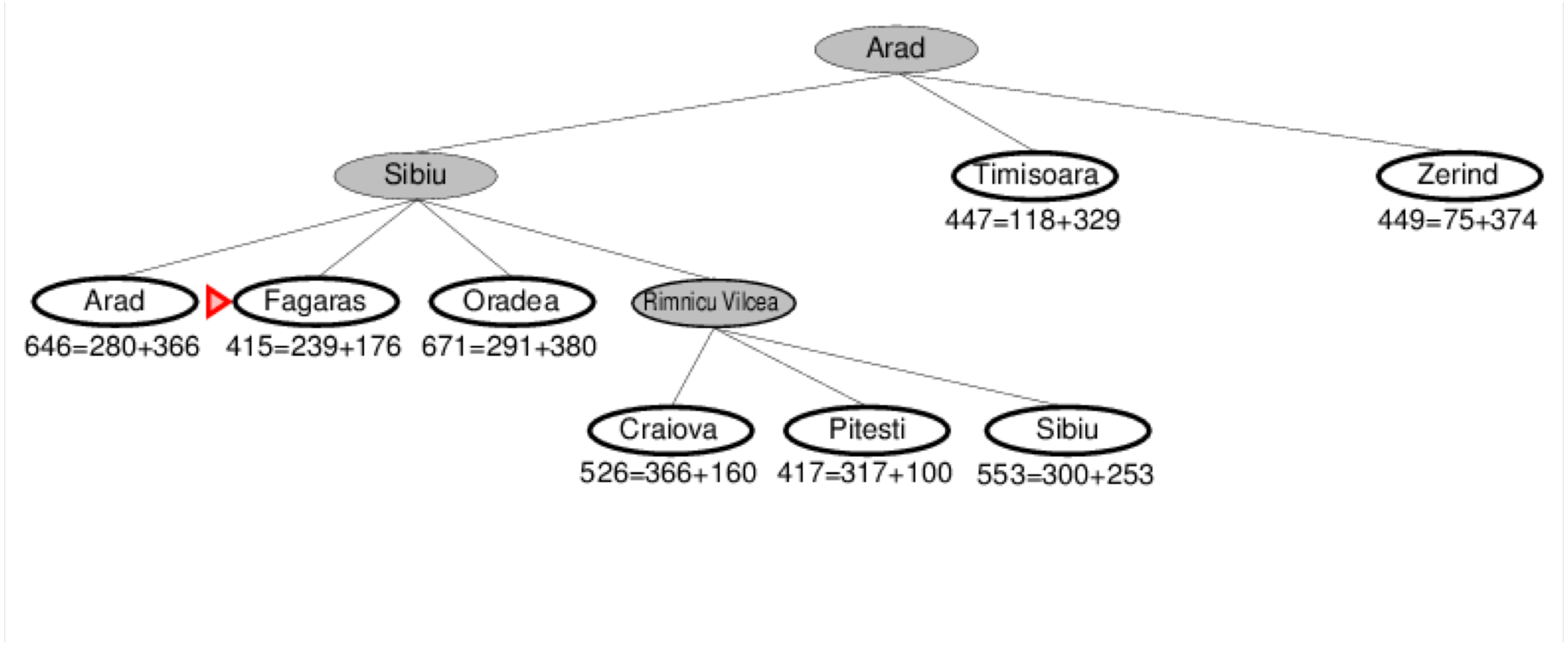
A* Search in Action



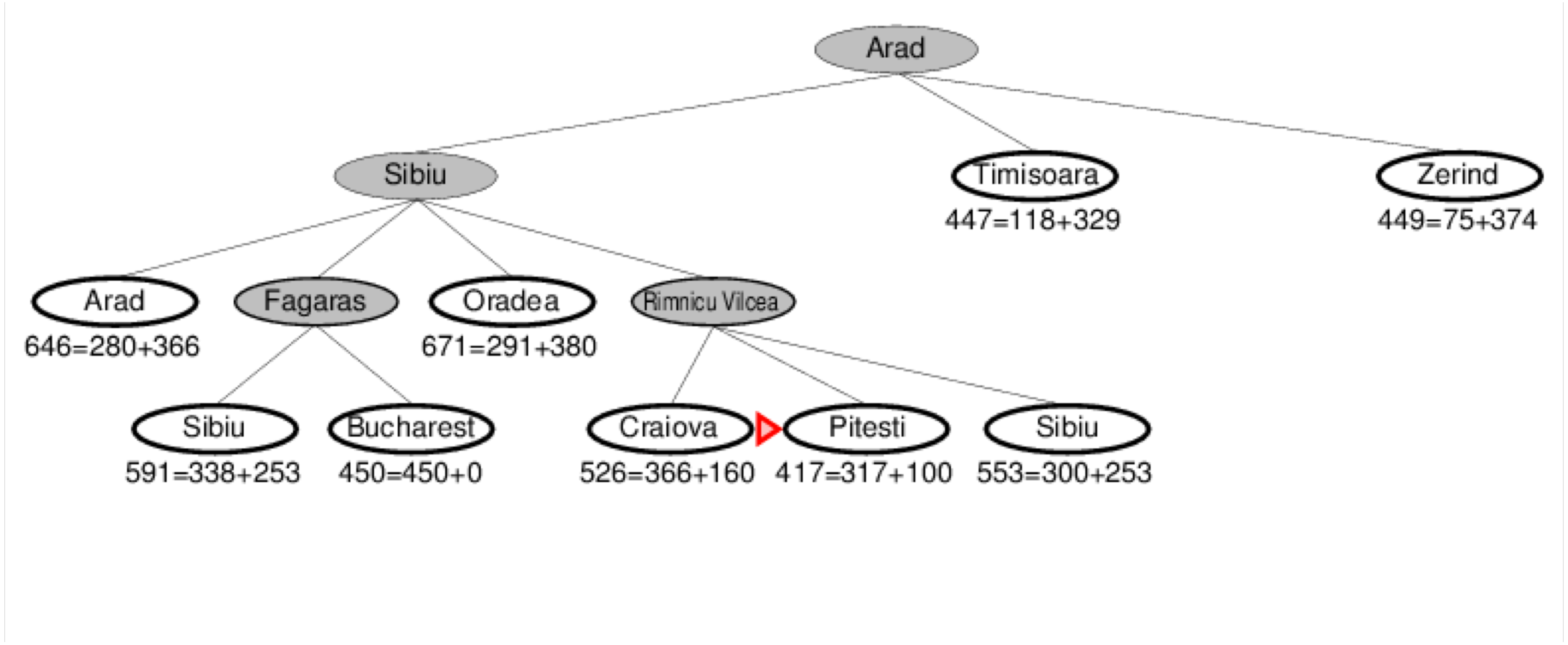
A* Search in Action



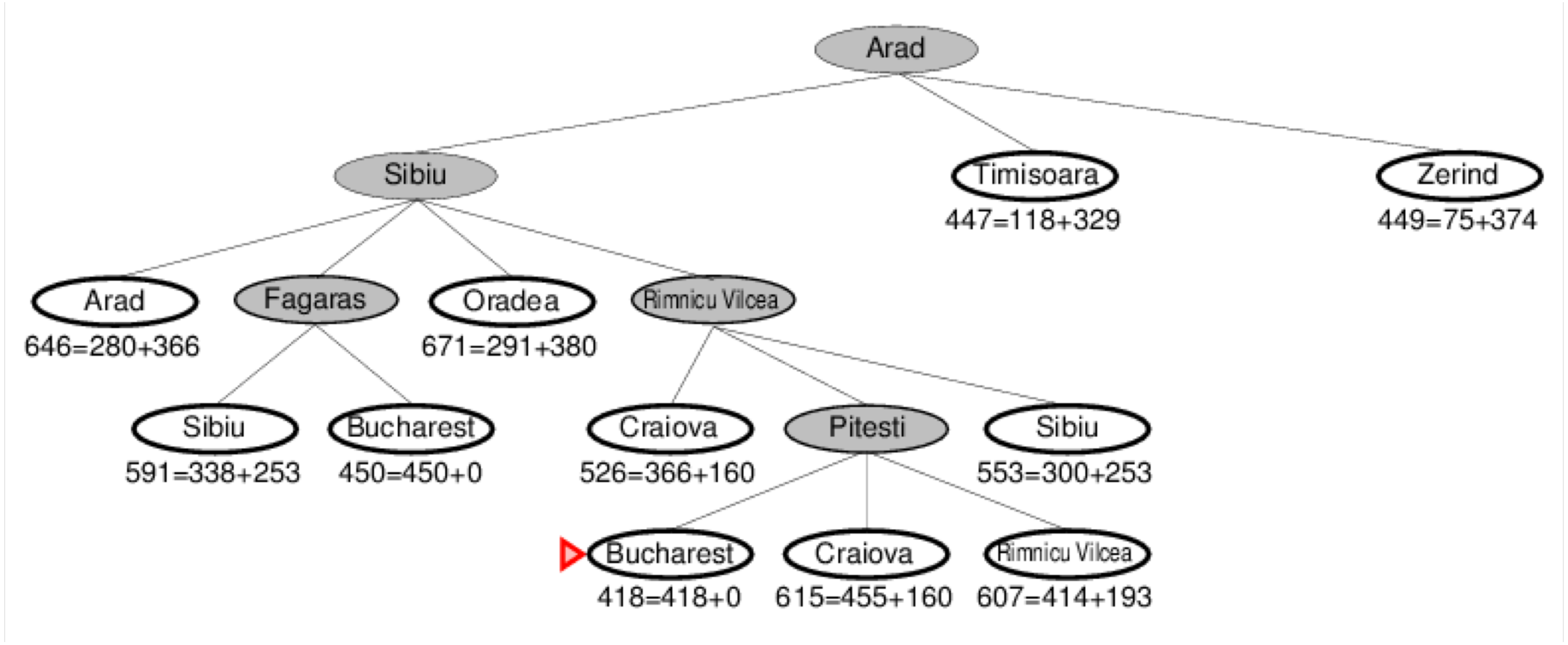
A* Search in Action



A* Search in Action



A* Search in Action



Optimality of A*

Tree-search version of A* is optimal if h is admissible

- Does not overestimate lowest cost from a vertex to the goal

Graph-search version additionally requires that h be consistent

- Estimated cost of reaching goal from a vertex n is not greater than cost to go from n to its successors and then the cost from them to the goal.
- Consistency is stronger, and it implies admissibility

Need to show:

Lemma 1: If *h* is consistent, then values of *f* along any path are nondecreasing

Lemma 2: If *h* is admissible, whenever A* selects a vertex v for expansion (extracts from the fringe), optimal path to v has been found

Proof of Lemma 1: Consistency -> Nondecreasing *f* along a Path

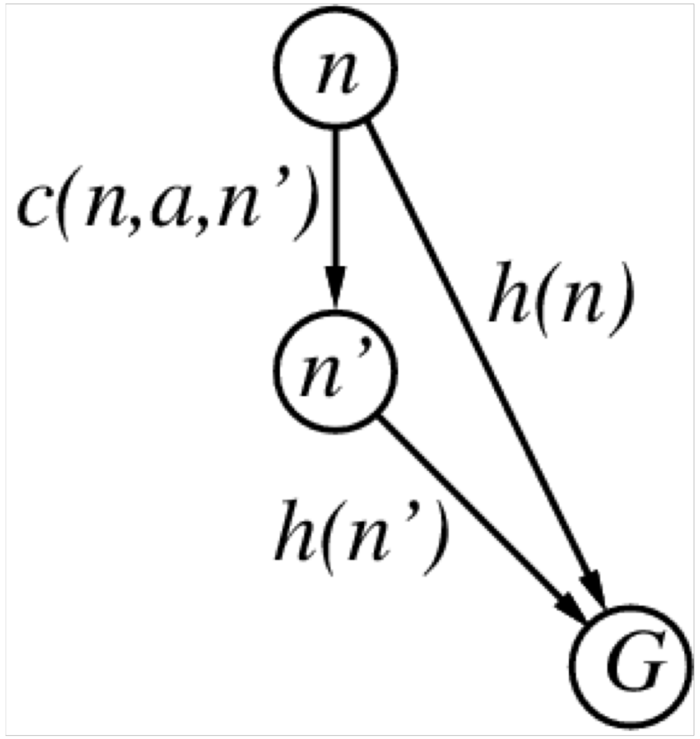
A heuristic is consistent if:

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

If *h* is consistent, we have:

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \end{aligned}$$

i.e., *f*(n) is nondecreasing along any path.



JAMES MADISON UNIVERSITY

Proof of Lemma 2: Consistency -> Admissibility

h(n): does not overestimate cost of lowest-cost path from *n* to *g*

- $h(n) \leq \delta(n, g)$

On the other hand

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

AND

- $h(n') \leq \delta(n', g)$

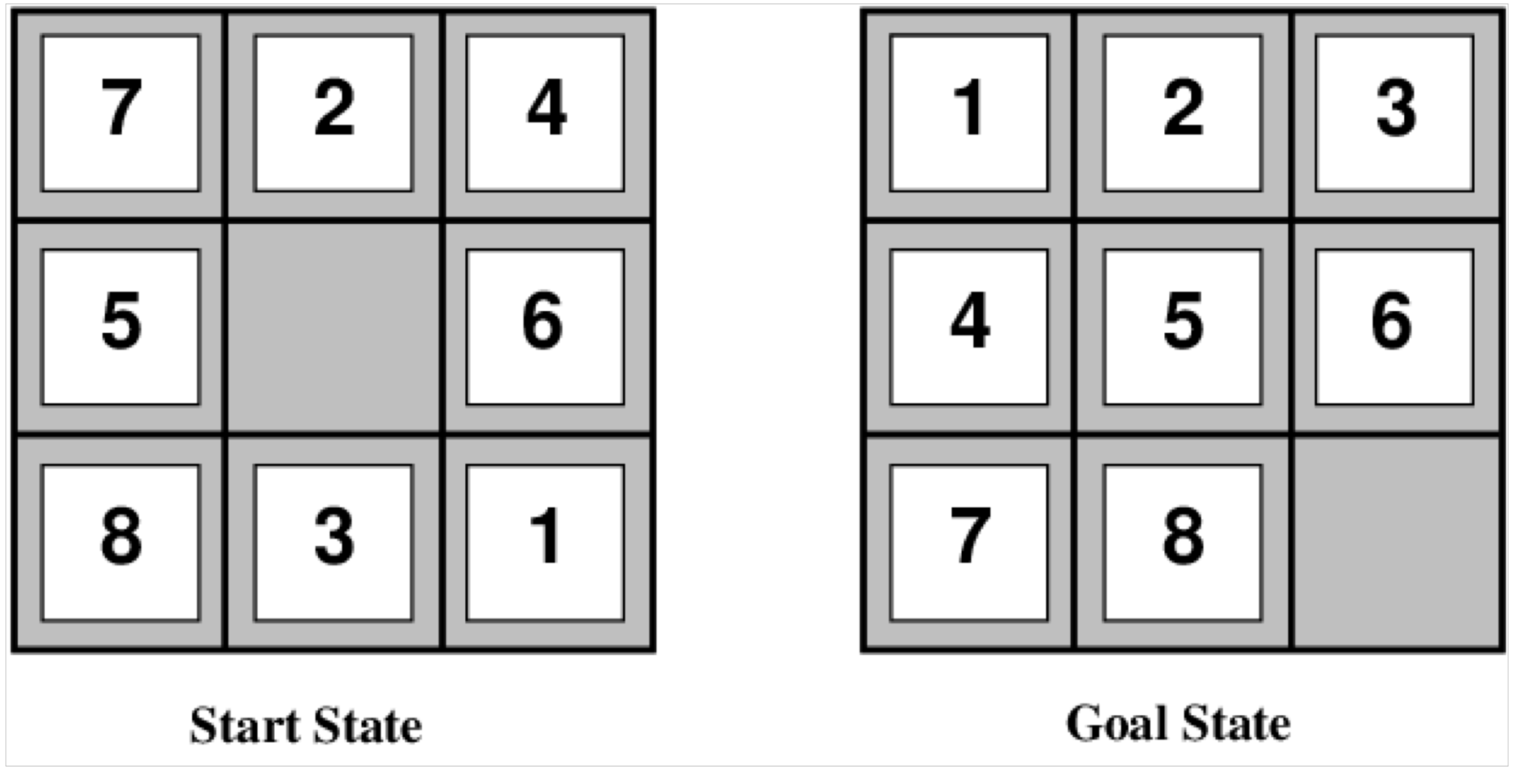
So

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

Admissible Heuristics

E.g., for the 8-puzzle

$h_1(v)$ = number of misplaced tiles



$h_1(S) = 6$

A* Performance

Depth (moves in optimal solution)	IDS (nodes)	A* (nodes)
14	3,473,941	
24	54,000,000,000	

A* Performance

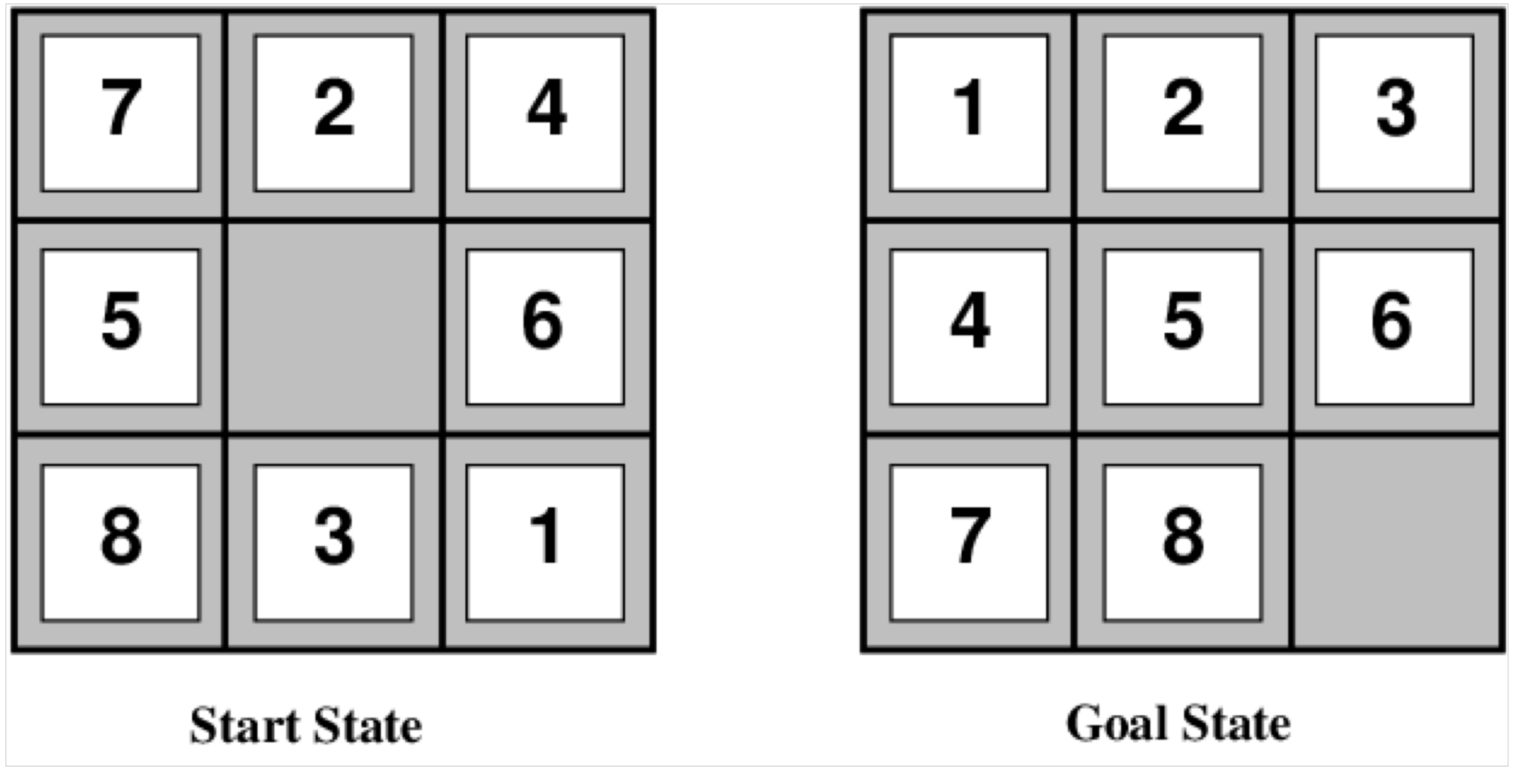
Depth (moves in optimal solution)	IDS (nodes)	A* (nodes)
14	3,473,941	539
24	54,000,000,000	39,135

Admissible Heuristics

E.g., for the 8-puzzle

$h_1(v)$ = number of misplaced tiles

$h_2(v)$ = total Manhattan distance



$h_1(S) =$ 6

$h_2(S) =$ 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14

JAMES MADISON UNIVERSITY.

A* Dominance

If $h_2(v) \geq h_1(v)$ for all *v* (both admissible)

Then h_2 ***dominates*** h_2 *and is better for search*

Depth (moves in optimal solution)	IDS (nodes)	A* (nodes) with h1	A*(nodes) with h2
14	3,473,941	539	113
24	54,000,000,000	39,135	1,641

Given any admissible heuristics h_a, h_b:

$$h(v) = \max(h_a(v), h_b(v))$$

Is also admissible and dominates h_a and h_b.

Technique for Heuristics – Relaxed Problem

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.

8 –puzzle:

- H1 – relax the problem so that the pieces can be just placed in their correct position in 1 move
- H2 – relax the problem so that the pieces can be placed in their correct positions directly counting each move (tiles can be moved to any adjacent square, not just where the blank space is located)

Key point: the optimal solution cost of the relax problem is no greater than the cost of the optimal solution to the real problem.

Summary of A* Search

Complete?:	Yes, unless there are infinitely many nodes with $f \leq f(G)$
Time?	Exponential in [path length x]
Space?	Keeps all generated nodes in memory (worse drawback than time)
Optimal?	Yes