CS 4700: Foundations of Artificial Intelligence

Fall 2017

Instructor: Prof. Haym Hirsh

Lecture 6

Today

- Informed Search (R&N Ch 3)

Thursday, February 16

- Informed Search (R&N Ch 3)
- Adversarial Search (T&N Ch 3)
- 4701 – last 15 minutes

4700 / 4701 TAs

PhD:

- Zikai Wen: zw385@cornell.edu (head TA)
- Cong Ding: cd564@cornell.edu

MS:

- Vaidehi Patel: vhp25@cornell.edu

MEng:

- Anant Agarwal: aa2387@cornell.edu
- Teddy Heidmann: ath55@cornell.edu
- Richie Henwood: rbh228@cornell.edu
- Taehoon Lee: tl353@cornell.edu
- Chandra Saarthak: sc2776@cornell.edu

Undergraduate:

- Rachel Baer: rcb353@cornell.edu
- Zihan Guo: zg89@cornell.edu
- Gi Yoon Han: gh336@cornell.edu
- Qiukai Lin: ql65@cornell.edu
- Ryan Picard: rjp264@cornell.edu
- Rachelle Tompkins: rst58@cornell.edu
- Matt White: mpw73@cornell.edu

Will appear on website with office hours shortly

Analysis: Iterative Deepening

Criterion	DFS	BFS	Iterative Deepening
Complete?	No	Yes	Yes
Optimal?	No	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$
Space?	O(bd)	$O(b^d)$	

Analysis: Iterative Deepening

$$1 + (1+b) + (1+b+b^2) + \cdots + (1+b+\cdots b^d)$$

Analysis: Iterative Deepening

$$1 + (1+b) + (1+b+b^2) + \cdots + (1+b+\cdots b^d)$$

$$= \sum_{i=0}^{0} b^i + \sum_{i=0}^{1} b^i + \sum_{i=0}^{2} b^i + \cdots + \sum_{i=0}^{d} b^i$$

Analysis: Iterative Deepening

$$\sum_{i=0}^{n} b^i = \frac{b^{n+1} - 1}{b - 1}$$

Analysis: Iterative Deepening

$$\sum_{i=0}^{n} b^i = \frac{b^{n+1} - 1}{b - 1}$$

$$\sum_{i=1}^{n} b^i = \frac{b^{n+1} - 1}{b - 1} - 1$$

Analysis: Iterative Deepening

$$1 + (1+b) + (1+b+b^2) + \cdots + (1+b+\cdots b^d)$$

$$= \sum_{i=0}^{0} b^i + \sum_{i=0}^{1} b^i + \sum_{i=0}^{2} b^i + \cdots + \sum_{i=0}^{d} b^i$$

$$= \frac{b^1 - 1}{b - 1} + \frac{b^2 - 1}{b - 1} + \frac{b^3 - 1}{b - 1} + \cdots + \frac{b^{d+1} - 1}{b - 1}$$

Analysis: Iterative Deepening

$$1 + (1+b) + (1+b+b^2) + \cdots + (1+b+\cdots b^d)$$

$$= \sum_{i=0}^{0} b^i + \sum_{i=0}^{1} b^i + \sum_{i=0}^{2} b^i + \cdots + \sum_{i=0}^{d} b^i$$

$$= \frac{b^1 - 1}{b-1} + \frac{b^2 - 1}{b-1} + \frac{b^2 - 1}{b-1} + \cdots + \frac{b^{d+1} - 1}{b-1}$$

$$= \frac{\sum_{i=1}^{d+1}(b^i - 1)}{b-1} = \frac{\sum_{i=1}^{d+1} b^i - \sum_{i=1}^{d+1} 1}{b-1} = \frac{\frac{b^{d+2}-1}{b-1} - 1 - (d+1)}{b-1} = O\left(\frac{b^{d+2}}{b^2}\right) = O(b^d)$$

Analysis: Iterative Deepening

Criterion	DFS	BFS	Iterative Deepening
Complete?	No	Yes	Yes
Optimal?	No	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$
Space?	O(bd)	$O(b^d)$	

Analysis: Iterative Deepening

Criterion	DFS	BFS	Iterative Deepening
Complete?	No	Yes	Yes
Optimal?	No	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$
Space?	O(bd)	$O(b^d)$	O(bd)

Informed Search

- Also called Heuristic Search
- Assume you have a “heuristic evaluation function” f(s) that measures the quality of a state
- Contrast with Uninformed Search
  - Uninformed search: methodical, but no information about s
  - Informed search: methodical, with constrained form of information about s

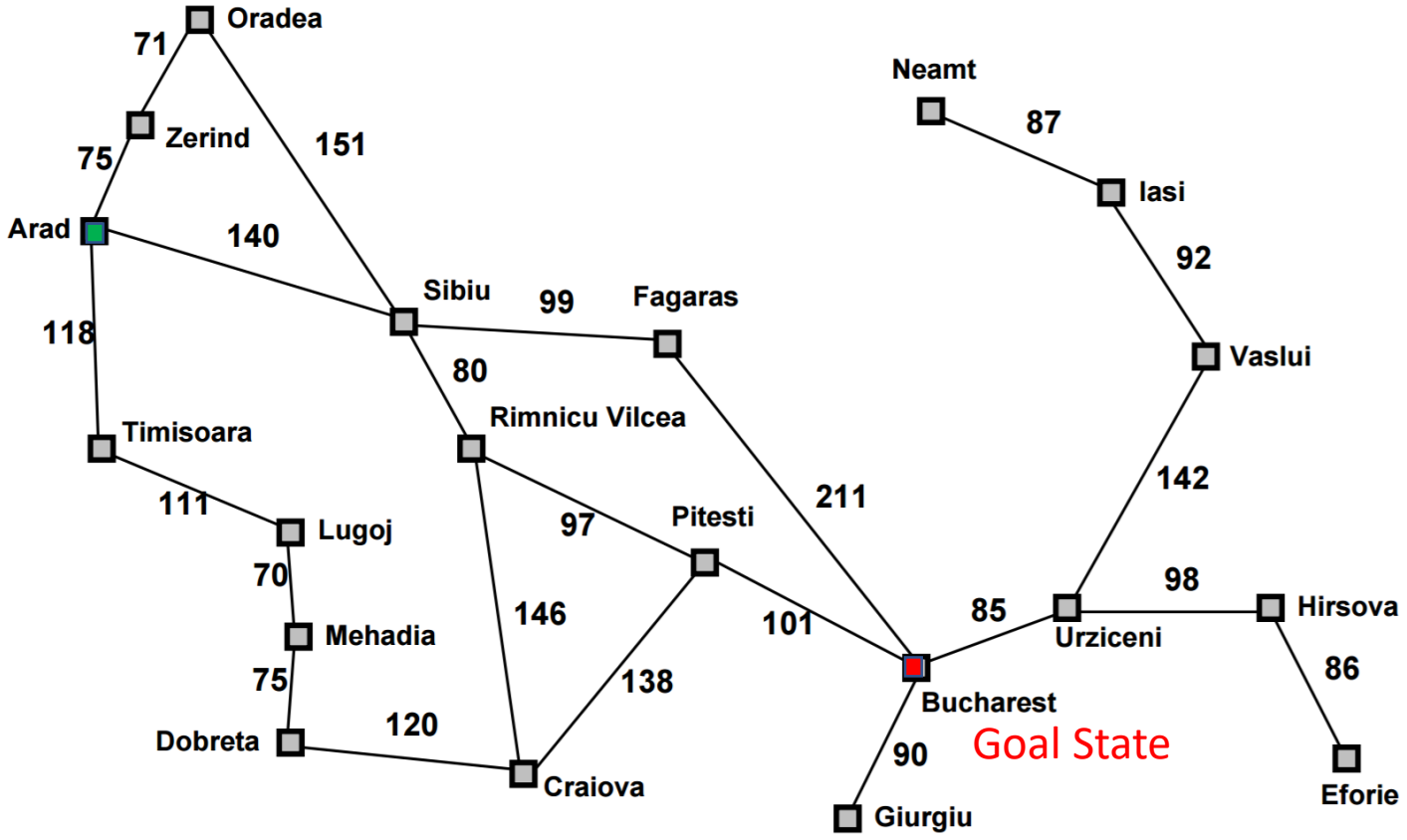
Heuristic Evaluation Functions

Special cases:

- f(s) = sum of costs from initial state to s
  - Operators can have variable costs
  - Seek least cost solution (= shortest solution when operators each cost 1)
  - Notation: f(s) = g(s)
- f(s) = estimate of cost from current state to s
  - Want fastest time to a goal
  - Notation: f(s) = h(s)
- f(s) = g(s) + h(s)
  - Estimate of cost of solution going through s

Heuristic Evaluation Function: Route Finding

Initial State



Heuristic Evaluation Function: Route Finding

- g(s) = sum of costs from Arad to a city s
- h(s) = distance between s and Bucharest on a map
- f(s) = estimate of distance from Arad to Bucharest through s

	h(s)=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

Heuristic Evaluation Function: 15 puzzle

- g(s) = number of moves of the blank space to s
- h(s) = number of out of place numbers in s

4	1	2	3
5	6	7	11
8	9		10
12	13	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- f(s) = estimate of solution length from initial state to goal via s

Heuristic Evaluation Function: 15 puzzle

- g(s) = number of moves of the blank space to s
- h(s) = number of out of place numbers in s

h(s) = 12

4	1	2	3
5	6	7	11
8	9		10
12	13	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- f(s) = estimate of solution length from initial state to goal via s

Heuristic Evaluation Function: 15 puzzle

- g(s) = number of moves of the blank space to s
- h(s) = sum of distances of times from correct place in s

4	1	2	3
5	6	7	11
8	9		10
12	13	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- f(s) = estimate of solution length from initial state to goal via s

Heuristic Evaluation Function: 15 puzzle

- g(s) = number of moves of the blank space to s
- h(s) = sum of distances of times from correct place in s

h(s) = 21

4	1	2	3
5	6	7	11
8	9		10
12	13	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- f(s) = estimate of solution length from initial state to goal via s

Best-First Search

```
BestFirstSearch(s,ops,open,closed) =
        If goal(s) then return(s);
          Else    open' ← open;
                  If s ∉ closed then
                          closed' ← closed + s;                    {include g(s) if necessary}
                          successors ← {};
                          For each o ∈ ops that applies to s
                                  successors ← successors + apply(o,s);   {include g(s) if necessary}
                          open' ← append(open',successors);
                     Else closed' ← closed;
                          update cost(s) if necessary;
                  If not(empty(open'))
                          s' ← best(open');                        {pick lowest according to f(s)}
                          open' ← remove(s',open');
                          BestFirstSearch(s',ops,open',closed')
                  Else return(FAIL)
Initial call: BestFirstSearch(initialstate,ops,{},{})
```

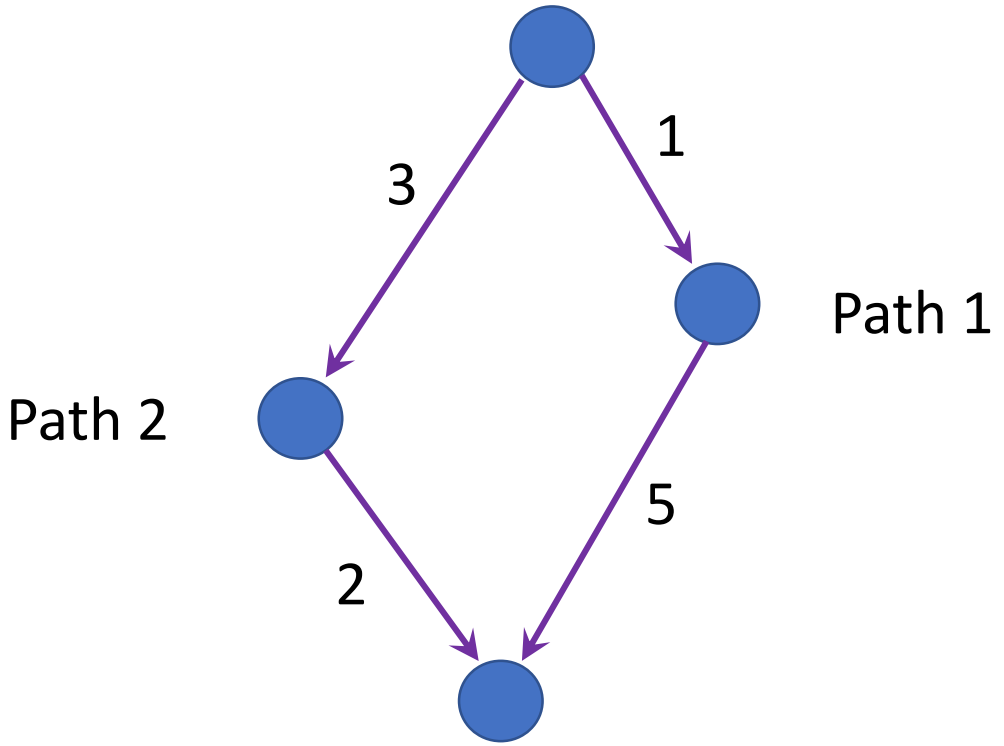
Best-First Search

```
BestFirst (s,ops,open,closed) =
    If goal(s) then return(s);
        Else  open' ← open;
              If s ∉ closed then
                  closed' ← closed + s;
                      {include g(s) if necessary}
                  successors ← {};
                  For each o ∈ ops that applies to s
                      s' ← apply(o,s);
                      If s' ∈ open then if new cost(s')
                           is lower, replace cost(s')
                      Else successors ← successors + s'
                  open' ← append(open',successors);
              Else    closed' ← closed;
        …
```

```
…
            If not(empty(open'))
                s' ← best(open');
                    {pick lowest according to f(s)}
                open' ← remove(s',open');
                     BestFirst(s',ops,open',closed')
                Else return(FAIL)
```

Initial call: BestFirst(initialstate,ops,{},{})

update cost(s)



Best-First Search

```
BestFirst (s,ops,open,closed) =
    If goal(s) then return(s);
        Else  open' ← open;
              If s ∉ closed then
                  closed' ← closed + s;
                      {include g(s) if necessary}
                  successors ← {};
                  For each o ∈ ops that applies to s
                      s' ← apply(o,s);
                      If s' ∈ open then if new cost(s')
                           is lower, replace cost(s')
                      Else successors ← successors + s'
                  open' ← append(open',successors);
              Else    closed' ← closed;
        …
```

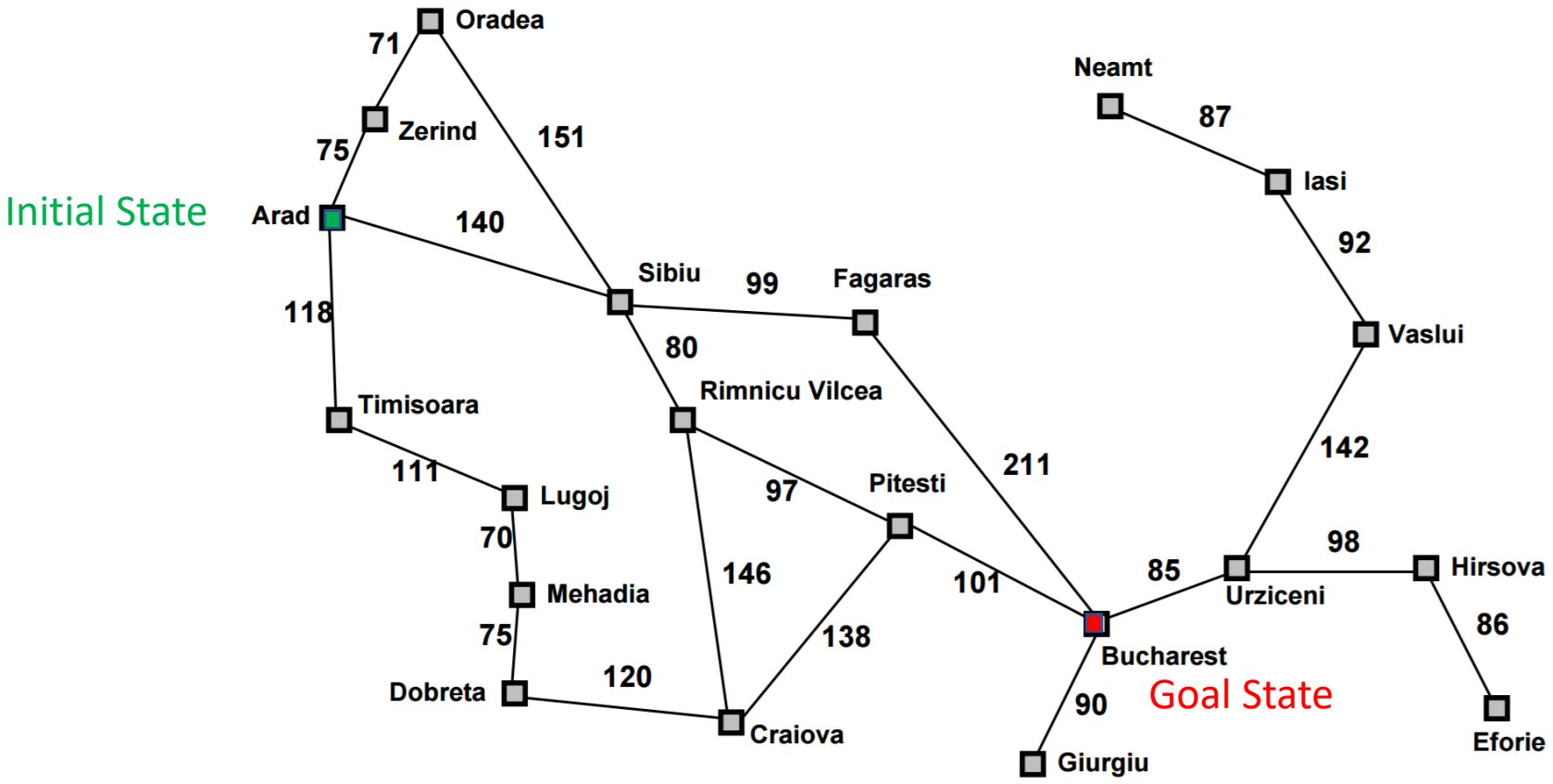
```
…
            If not(empty(open'))
                s' ← best(open');
                    {pick lowest according to f(s)}
                open' ← remove(s',open');
                     BestFirst(s',ops,open',closed')
                Else return(FAIL)
```

Initial call: BestFirst(initialstate,ops,{},{})

Special Cases of Best-First Search

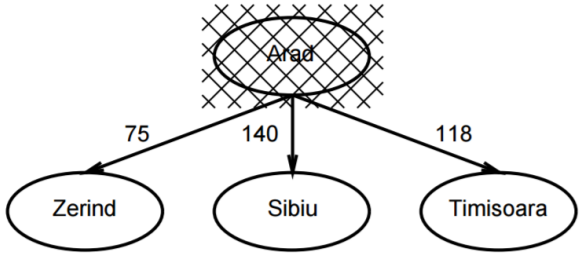
- f(s) = g(s): Uniform Cost Search
- f(s) = g(s) + h(s): A* Search

Uniform Cost Search

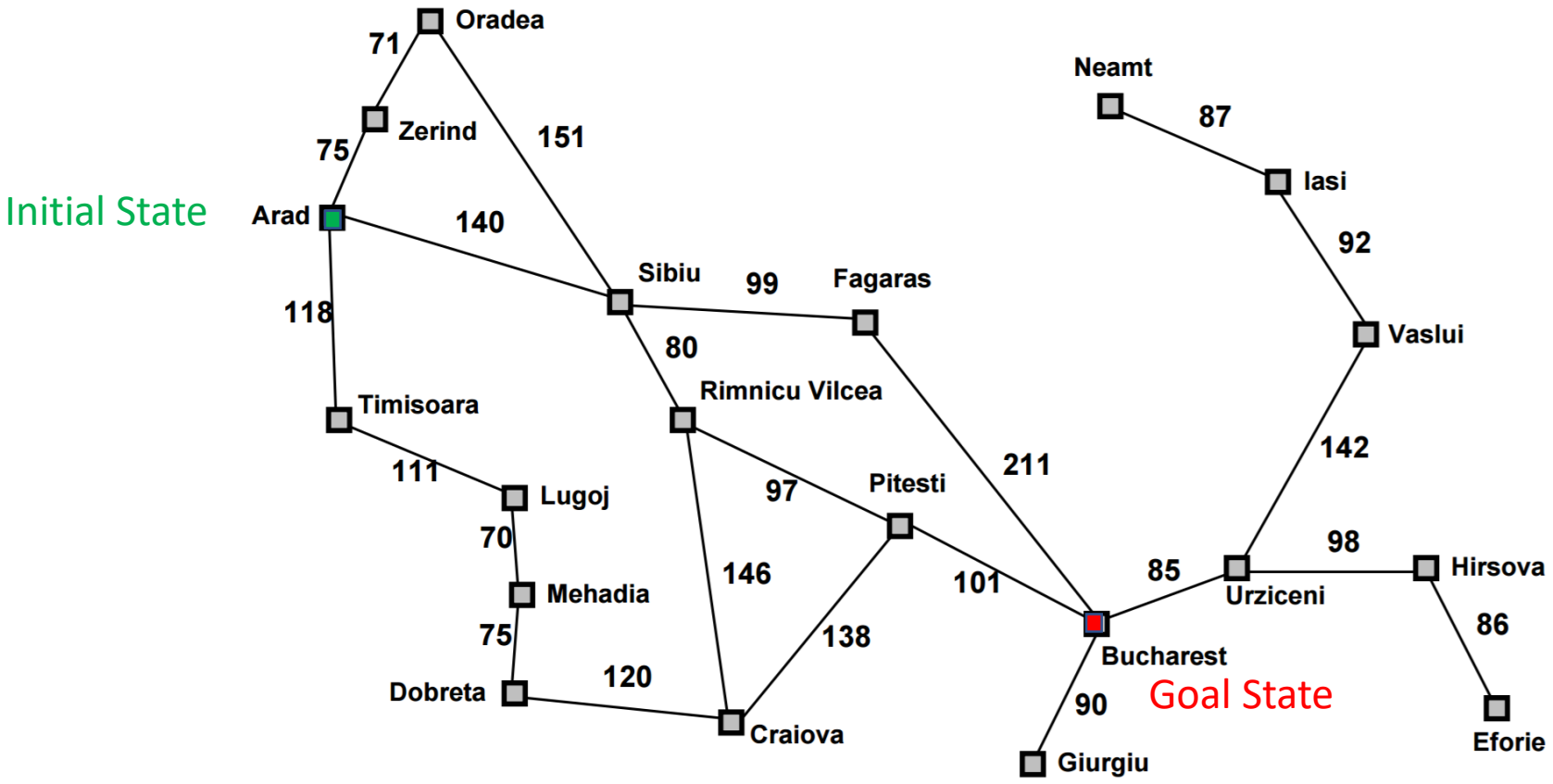


Uniform Cost Search

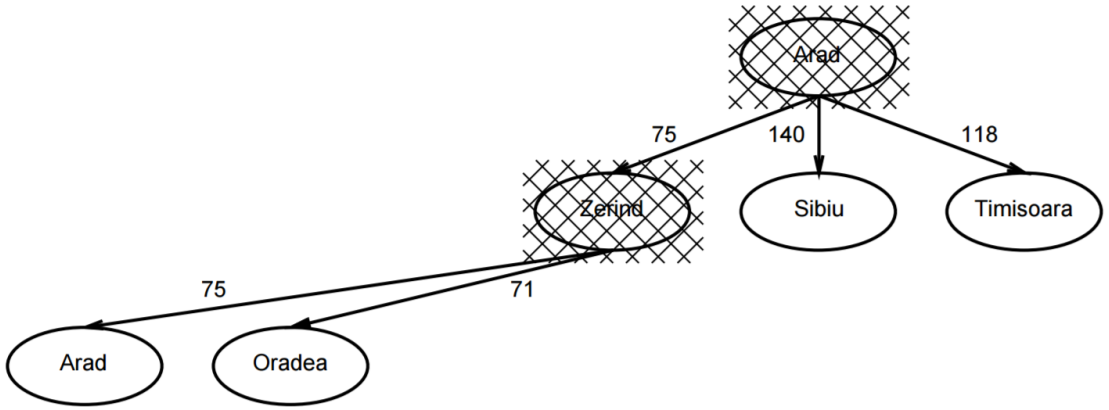
-



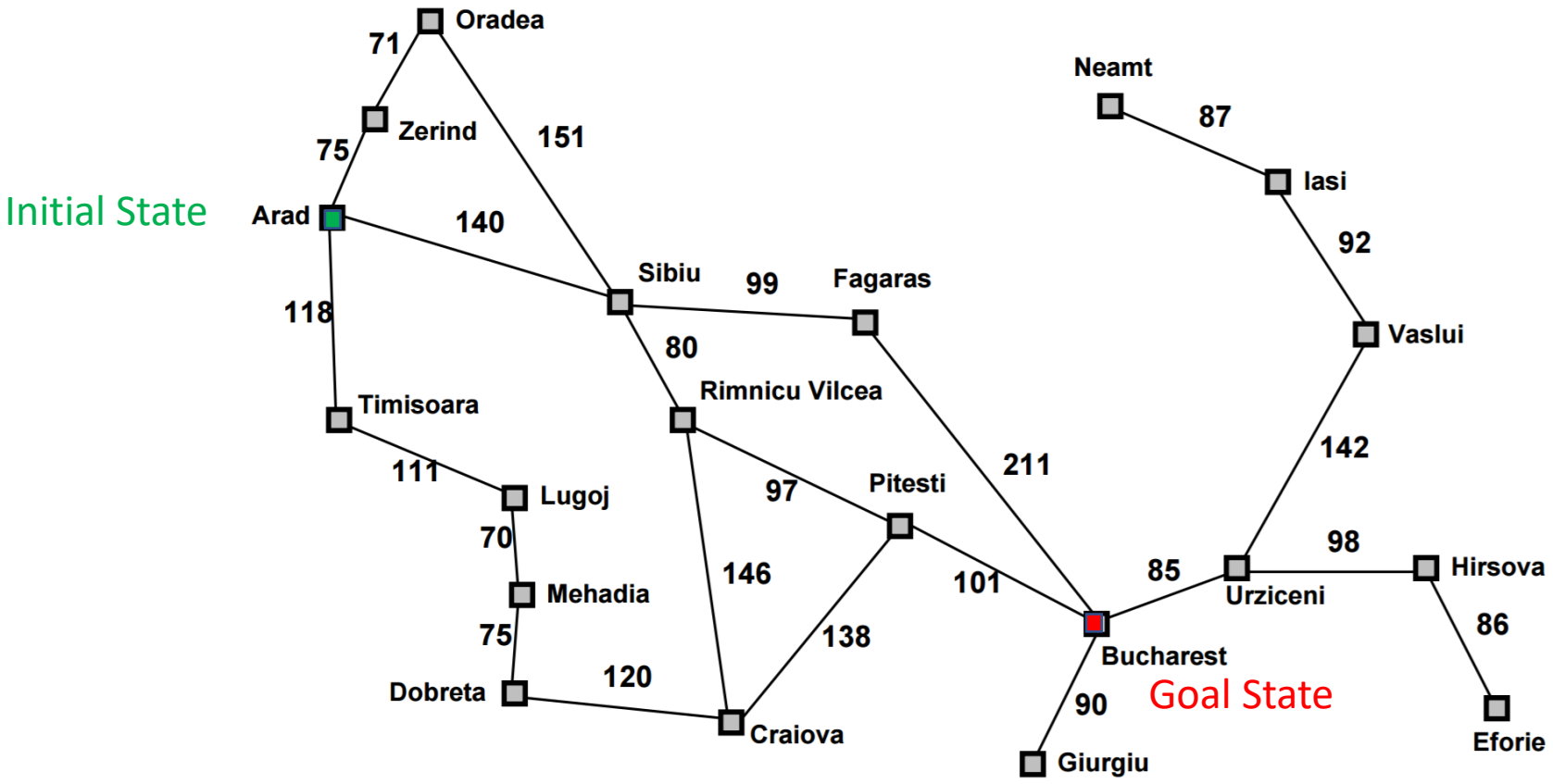
Uniform Cost Search



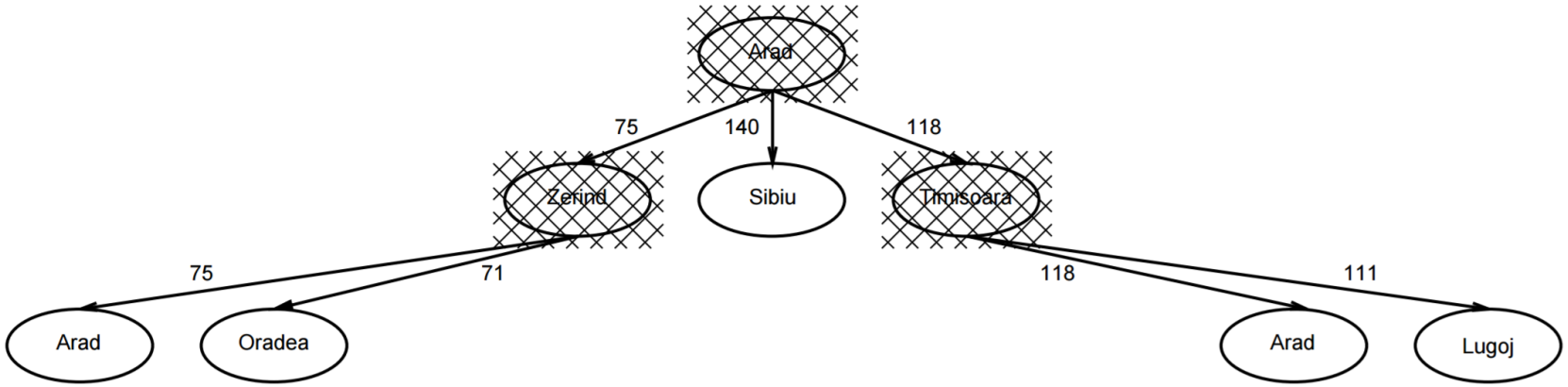
Uniform Cost Search



Uniform Cost Search



Uniform Cost Search



Analysis: Uniform Cost Search

Criterion	DFS	BFS	Iterative Deepening	Uniform Cost Search
Complete?	No	Yes	Yes	
Optimal?	No	Yes	Yes	
Time?	∞	$O(b^d)$	$O(b^d)$	
Space?	O(bd)	$O(b^d)$	O(bd)	

Analysis: Uniform Cost Search

Criterion	DFS	BFS	Iterative Deepening	Uniform Cost Search
Complete?	No	Yes	Yes	Yes
Optimal?	No	Yes	Yes	
Time?	∞	$O(b^d)$	$O(b^d)$	
Space?	O(bd)	$O(b^d)$	O(bd)	

Analysis: Uniform Cost Search

Criterion	DFS	BFS	Iterative Deepening	Uniform Cost Search
Complete?	No	Yes	Yes	Yes
Optimal?	No	Yes	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$	
Space?	O(bd)	$O(b^d)$	O(bd)	

Analysis: Uniform Cost Search

Criterion	DFS	BFS	Iterative Deepening	Uniform Cost Search
Complete?	No	Yes	Yes	Yes
Optimal?	No	Yes	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$	$O(b^d)$
Space?	O(bd)	$O(b^d)$	O(bd)	

Analysis: Uniform Cost Search

Criterion	DFS	BFS	Iterative Deepening	Uniform Cost Search
Complete?	No	Yes	Yes	Yes
Optimal?	No	Yes	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^d)$	$O(b^d)$
Space?	O(bd)	$O(b^d)$	O(bd)	$O(b^d)$