Queen Mary
University of London

Problem Solving and Search

Simon Dixon
Queen Mary University of London

What is problem solving?

Queen Mary University of London

- An agent can act by
  - establishing *goals* that result in solving a *problem*
  - considering *sequences of actions* that might achieve those goals
- To automate this, we need to *formulate the problem*
- The process of considering action sequences within a problem formulation is called *search*
- This part of the module is about
  - problem-solving agents
  - problem types
  - problem formulation
  - example problems
  - search strategies

An Agent on Holiday

Queen Mary University of London

- Example
  - *An agent is in Arad, Romania, at the end of a touring holiday. It has a ticket to fly out of Bucharest the following day. The ticket is non-refundable, the agent's visa is about to expire and there are no seats available on other flights for six weeks.*
- The agent's performance measure (in decreasing importance) includes
  - avoidance of arrest and/or deportation
  - cost of ticket
  - seeing more sights
  - improvement of suntan
- Lots of actions are possible to score highly on the performance measure, but the seriousness of the agent's situation means that the agent should adopt the *goal* of getting to Bucharest
  - Agent can reject any actions that fail to meet (or move towards) this goal

A problem-solving agent

Queen Mary University of London

- *Problem formulation* is the analysis of possible states and actions
  - Knowledge representation: *what* to represent and *how* to represent it
  - Initial state, goal state/test, actions (operators) and their effect on the state
- *Goals* help to organise behaviour
  - A goal is a desired state of (part of) the world
    - it does not usually specify the state of the *whole* world
    - alternatively, a goal can be thought of as the set of world states in which it is satisfied
- *Actions* cause transitions between *world states*
  - The agent has to work out which action(s) will get it to the goal state
- The agent needs to consider possible actions and the resulting states
    - put left foot forward 18 inches?? get cab to next town??

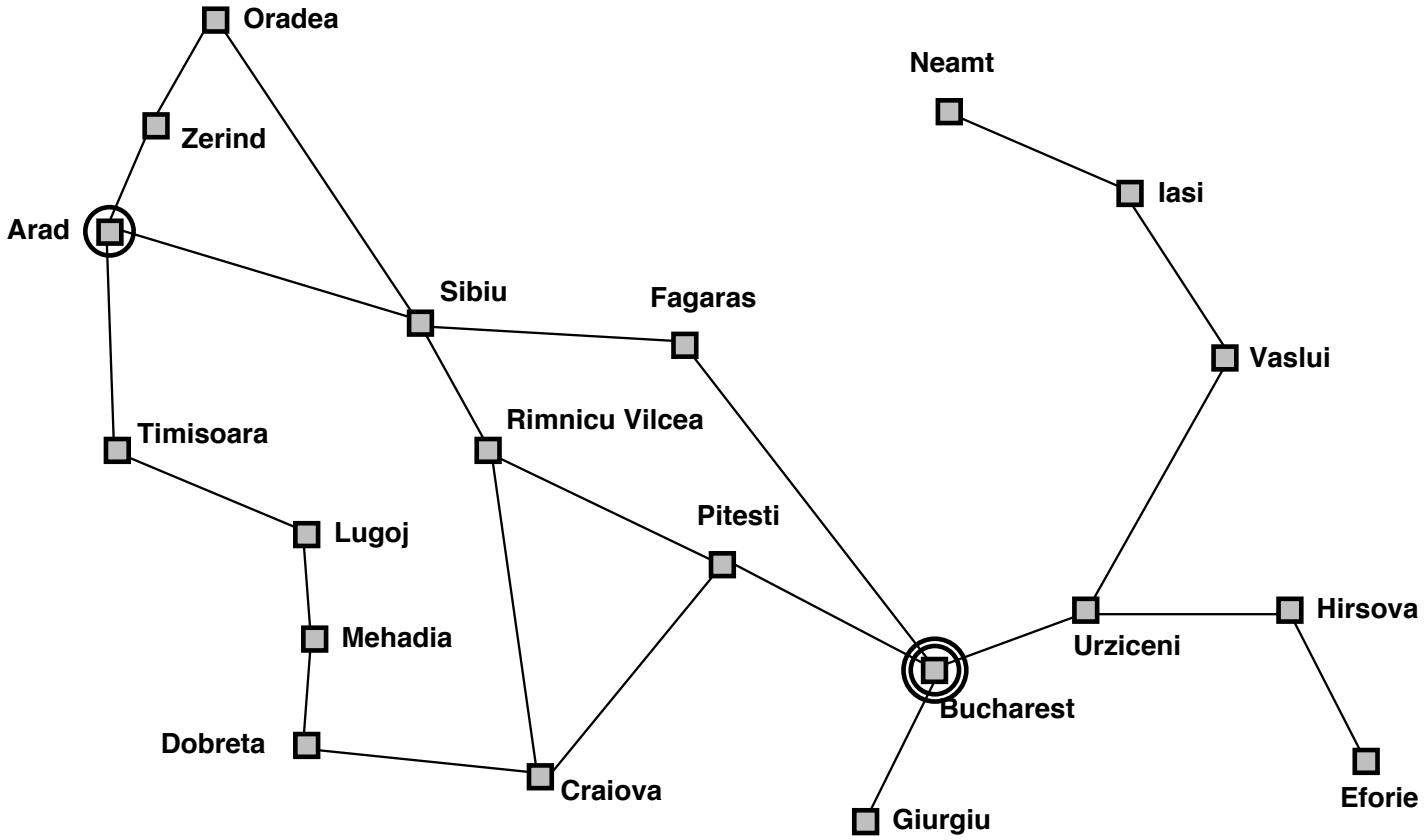
A problem-solving agent

Queen Mary University of London

- Our holiday agent has adopted the goal of driving to Bucharest
- There are three roads out of Arad: to Sibiu, Timisoara and Zerind
  - None of these reaches the goal. Which road should the agent follow?
- Without additional knowledge about Romania, the agent cannot choose the best road to take: it can only make a random choice
- However, if the agent has a map of Romania, it knows which states it can get to and which actions it can perform

Exercise: route planning on a map

Queen Mary University of London



- Find a route from Arad to Bucharest.
  - Is this the optimal route?
  - How did you go about finding the route?

A Problem-Solving Agent

Queen Mary University of London

- The agent can use the map information
  - consider *subsequent stages* of a hypothetical journey through each of the 3 nearest towns, to try to find a route that eventually leads to Bucharest
- Once it has found a path on the map from Arad to Bucharest, it can achieve its goal locally by driving to each town on the route
- More generally, with several immediate options of unknown value, the agent can decide what to do by *searching*
  - assessing different possible sequences of actions that lead to states of known *utility* (goal states)
  - choosing the best one

Design Process

Queen Mary University of London

- To design a problem-solving program
  - FORMULATE: Formulate the problem, starting conditions and goal
  - SEARCH: Use search algorithm to find action sequence that realises goal from starting conditions
  - EXECUTE: Use solution to guide actions

THE ART OF AI PROBLEM SOLVING IS DECIDING WHAT GOES INTO THE DESCRIPTION OF THE OPERATORS AND STATES, AND WHAT IS LEFT OUT

Well-defined problems

Queen Mary University of London

- Basic problem elements are states and actions
- Formally, we need
  - *initial state*: the world the agent starts in
  - *operators*: describe actions in terms of mappings (functions) from states to states
  - *goal state*: the world that the agent is trying to be in
- The initial state and operators define the *state space* of the problem
  - the set of all states reachable from the initial state and any sequence of actions
- To determine when the goal state is reached, we need
  - *goal test*: a boolean function over states, true if input is the goal state

Formulating the Farmer's Dilemma

Queen Mary University of London

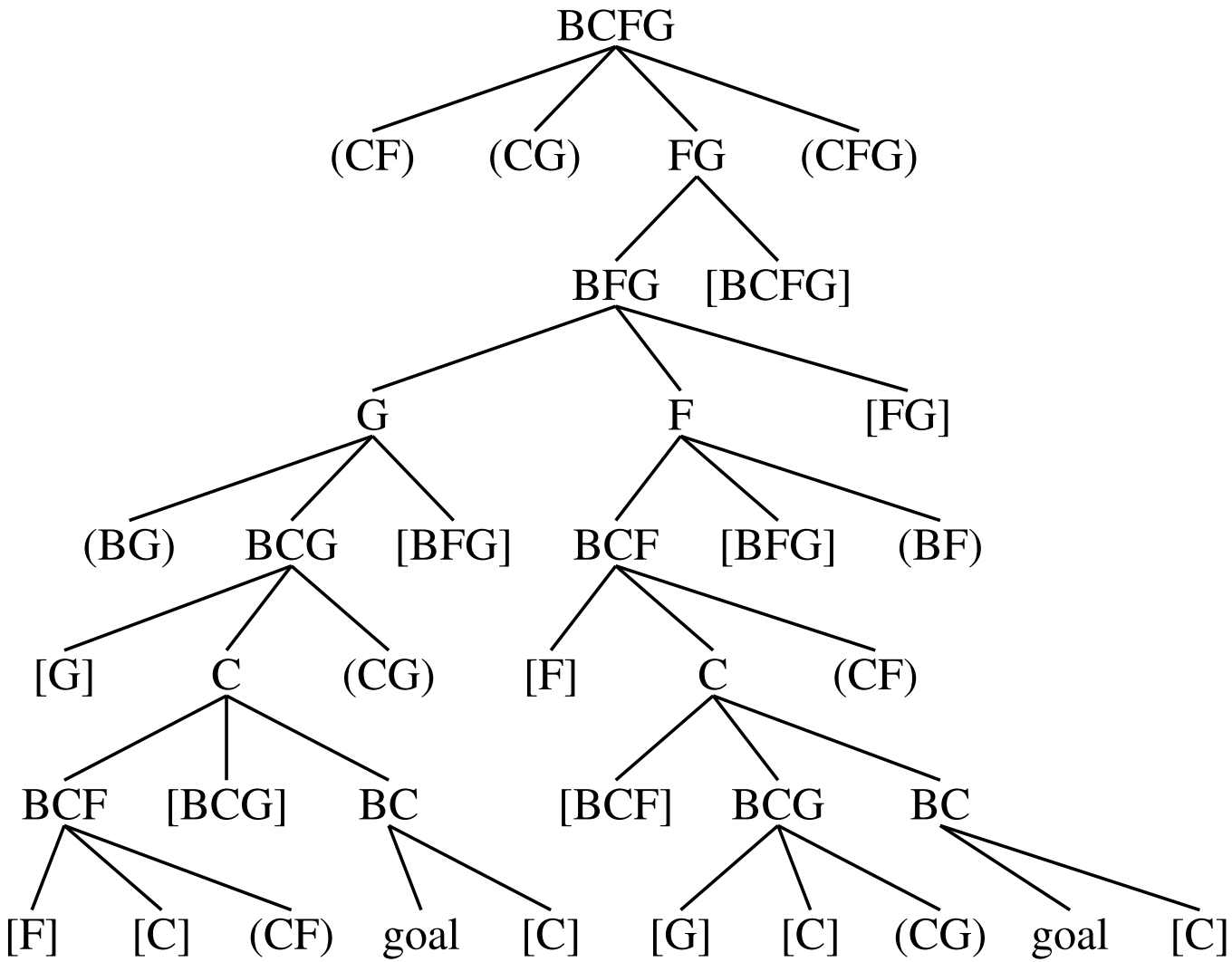
We can represent the state by the set of objects on the left bank:

- General state: LeftBank $\subseteq$ {Boat, Chicken, Fox, Grain}
  - Note: the farmer is always with the boat (not represented separately)
- Initial state: LeftBank = {Boat, Chicken, Fox, Grain}
- Operators: CrossRight(x) represents the farmer crossing the river to the right, taking x $\in$ {Nothing, Chicken, Fox, Grain} with him
  - CrossRight(x) is legal if Boat $\in$ LeftBank and x $\in$ LeftBank or x=Nothing
  - CrossRight(x) results in: LeftBank := LeftBank - {Boat, x}
  - Check that {Chicken, Fox} $\notin$ LeftBank and {Chicken, Grain} $\notin$ LeftBank
  - CrossLeft(x) is defined similarly
- Goal test: LeftBank == {}
- Path cost: Number of operations performed

Farmer's Dilemma: Search Space

Queen Mary University of London

Each node represents a state; the children of a node are the states that can be reached from the parent state. Repeated states are in brackets, illegal states in parentheses. The initial state is represented by the root of the tree, and two of the leaves represent goal states.



Search Algorithms

Queen Mary University of London

- A search algorithm takes a *problem* as input, *formulated* as
  - a description of the world (the initial state)
  - a set of possible *actions* (or operators) to be applied
  - a set of *goals* to be achieved (can be expressed as goal tests)
- It returns a solution in the form of an *action sequence* or *plan*
- Once a solution is found, the chosen actions can be carried out
  - the *execution phase*
- In the agent context, this becomes complicated because of interaction between agents and other dynamic things in the world
  - so we consider the search on its own first

Searching for solutions

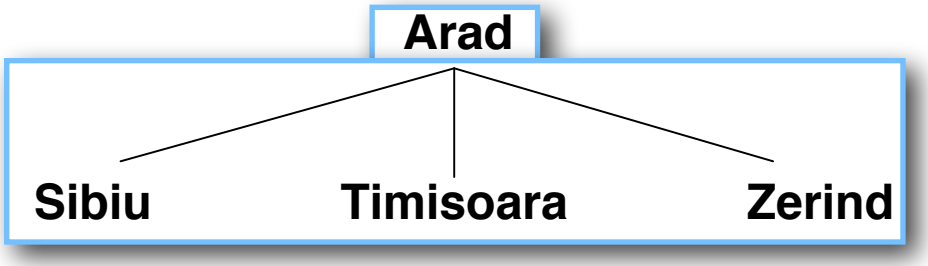
Queen Mary University of London

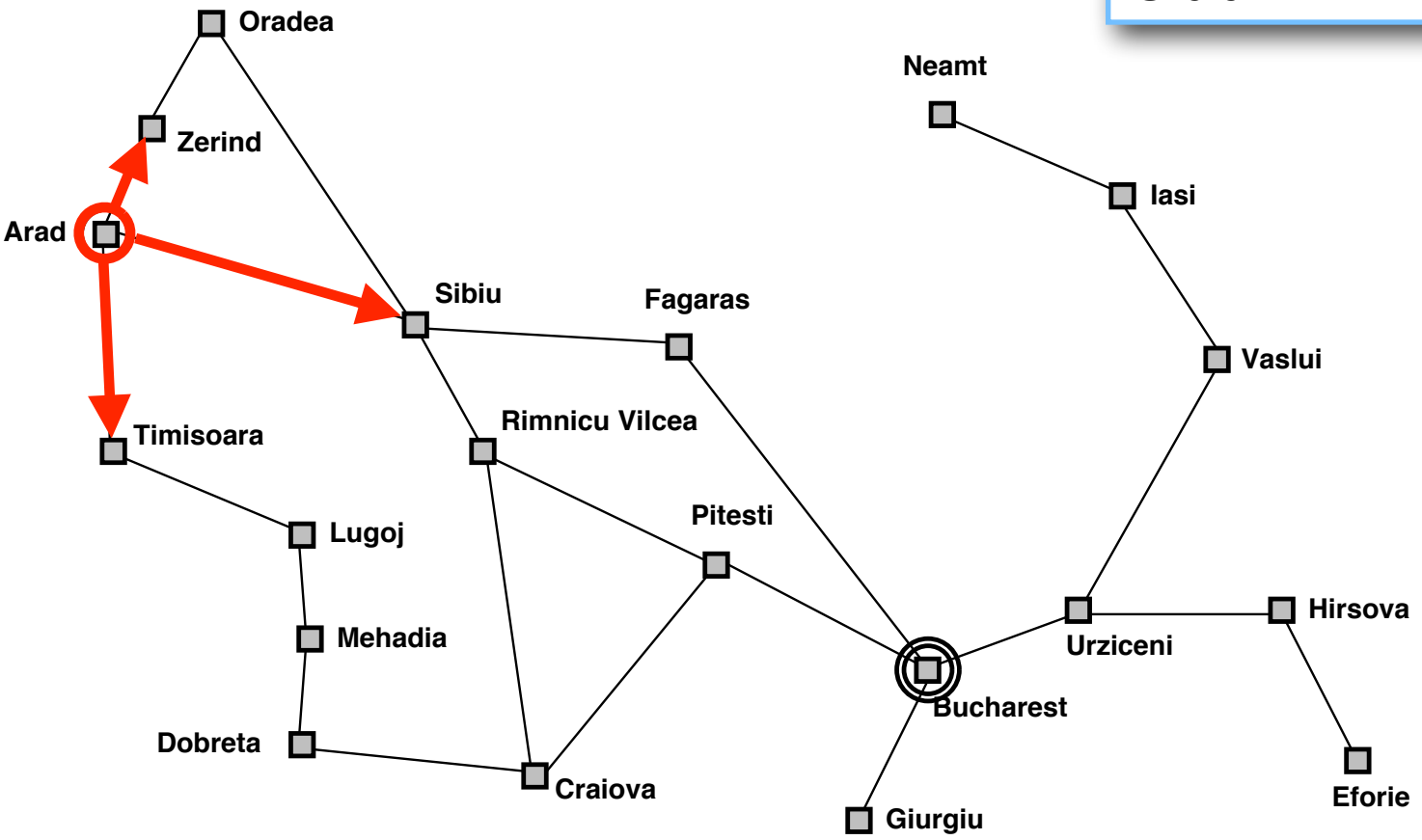
- A solution is a sequence of operators leading from start to goal state
- Given a problem formulation, we have to map out the state space
  - while we are searching the state space it's sometimes called the *search space*
- Generate action sequences, using an *agenda*:
  - start at initial state
  - check whether in goal state
    - if yes, we're done
  - otherwise, consider other states by applying the operators to the present state to generate a new set of states
    - This is called *expanding* the state

Search Trees

Queen Mary University of London

- It's helpful to think of the state space as structured into a *tree*
  - actions correspond with *arcs*
  - states correspond with *nodes*

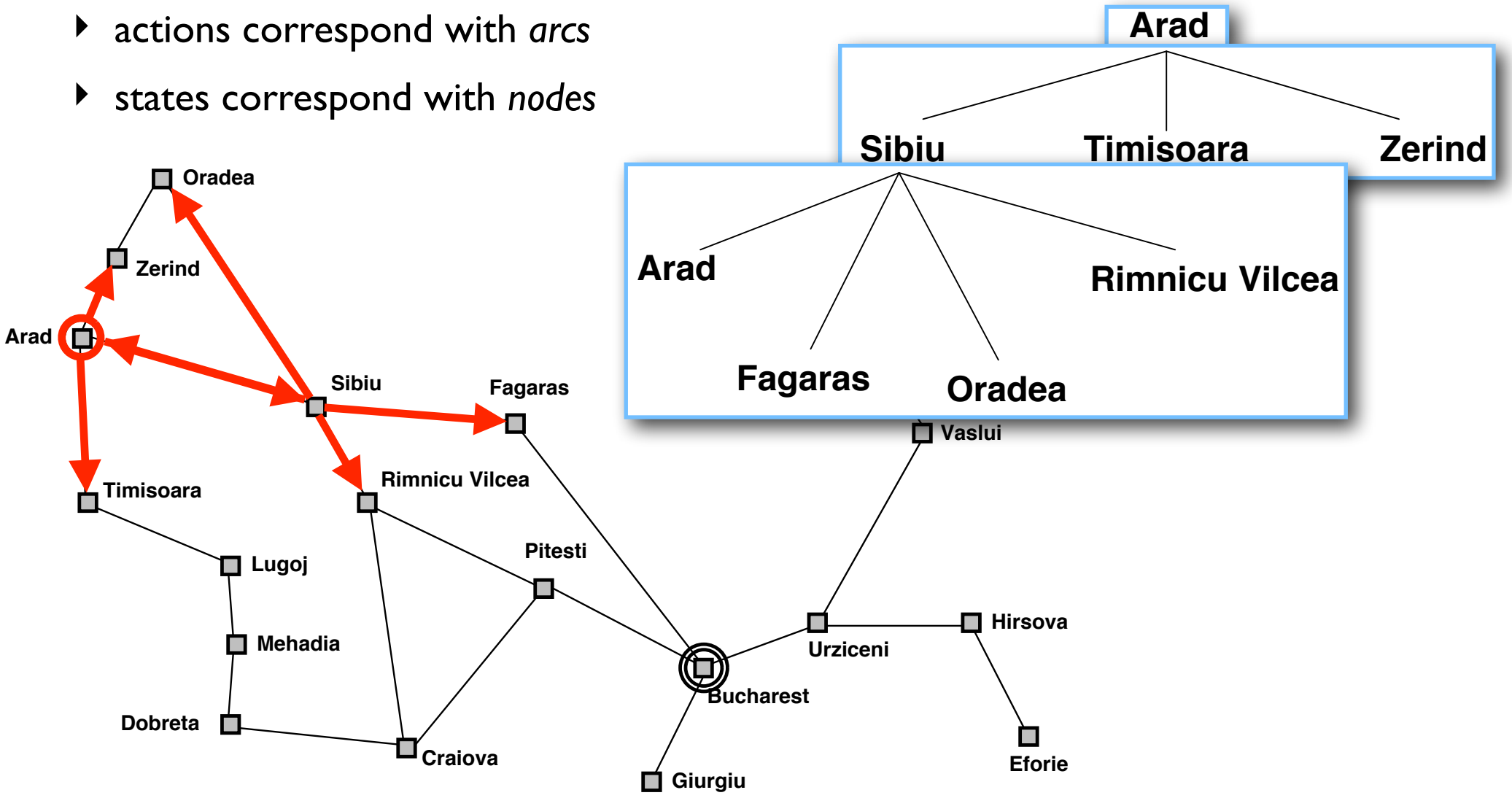




Search Trees

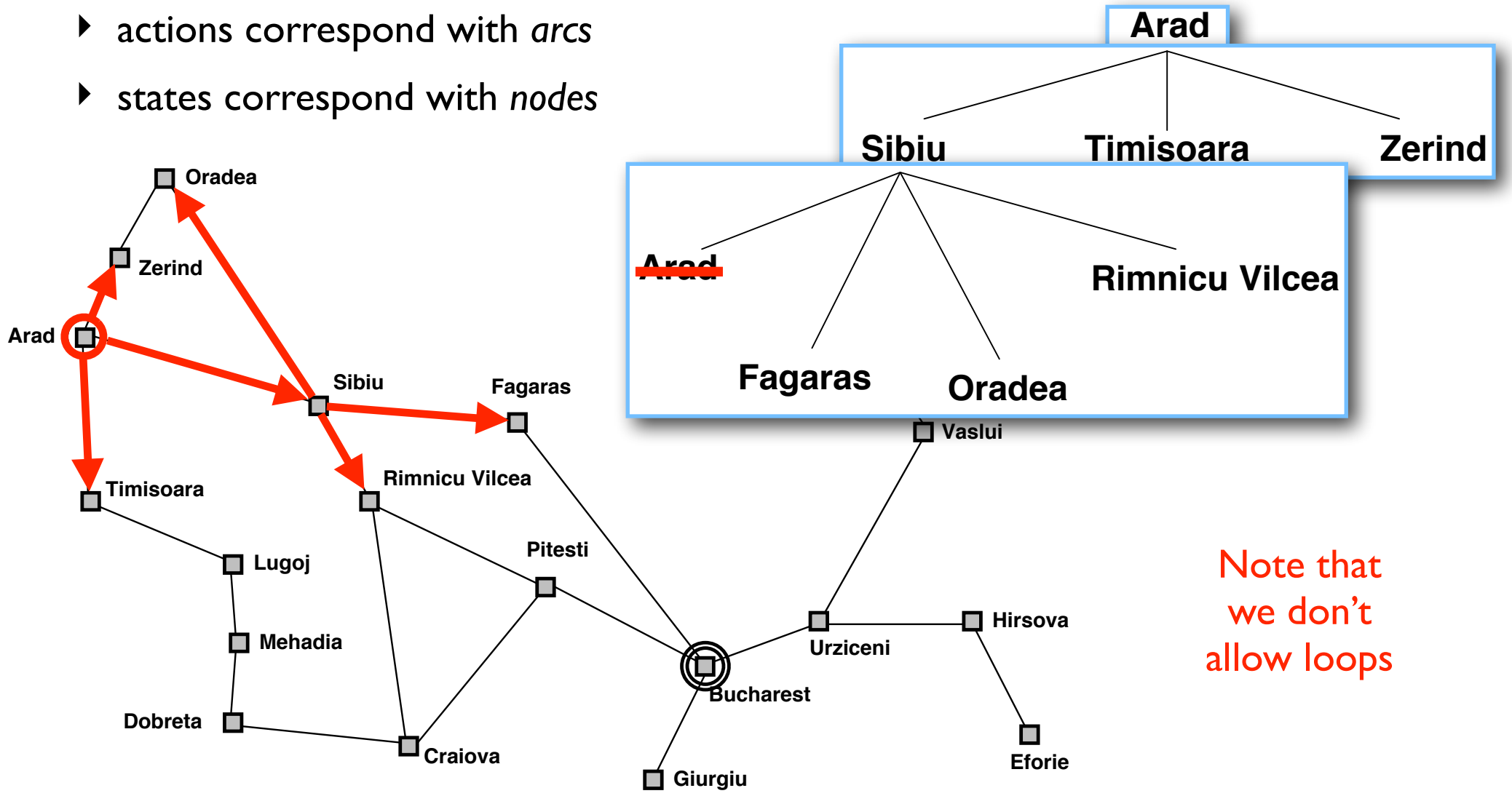
Queen Mary University of London

- It's helpful to think of the state space as structured into a *tree*
  - actions correspond with *arcs*
  - states correspond with *nodes*



Search Trees

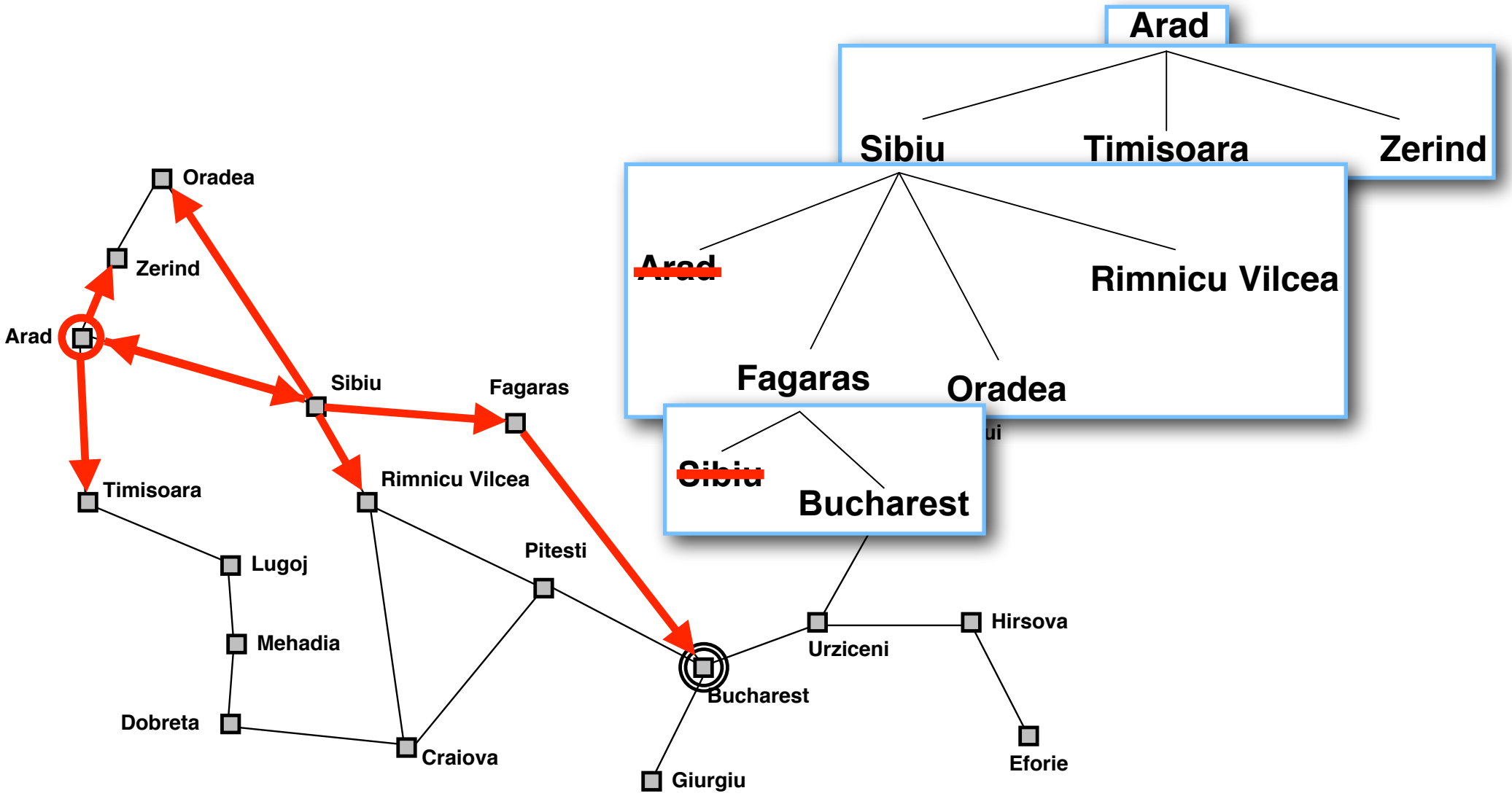
- It's helpful to think of the state space as structured into a *tree*
  - actions correspond with *arcs*
  - states correspond with *nodes*



Note that we don't allow loops

Search Trees

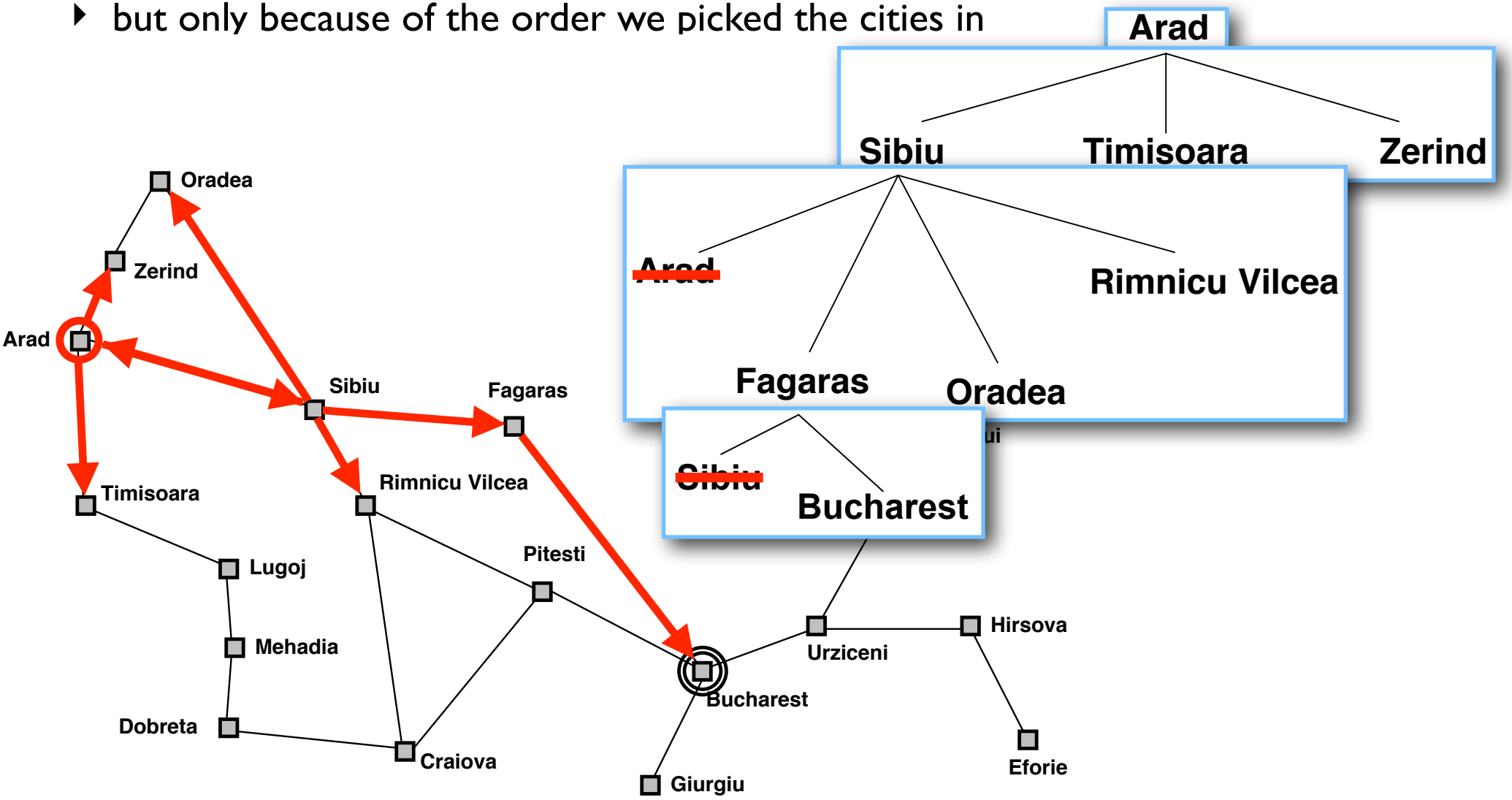
Queen Mary University of London



Search Trees

Queen Mary University of London

- Searching this way takes us to Bucharest in 3 steps
  - but only because of the order we picked the cities in



Clear thinking: state ≠ node!

Queen Mary University of London

- States of the world and nodes in the search tree are not the same thing
  - A state is (a representation of) a problem configuration
  - A node is a data structure used by a search algorithm
    - constitutes part of a search tree
    - includes parent, action, children, depth (for IDS), path cost (for UCS)
  - States do not formally have parents, actions, children, depth, or path cost!
  - There may be more than one way to reach the same state
- The Expand function creates new *nodes*, filling in the various fields and using the operators of the problem formulation to approximate the corresponding *states*

Search Algorithm Framework

Queen Mary University of London

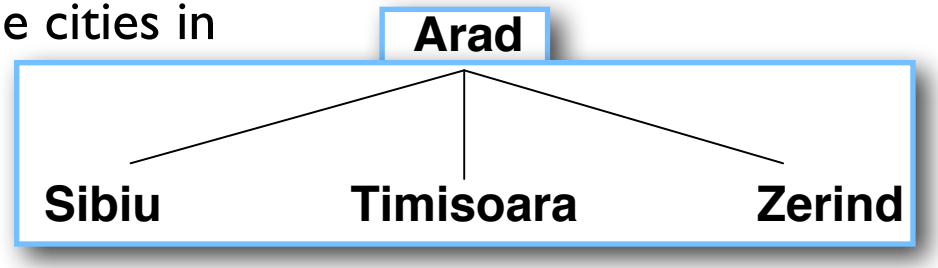
```
function AgendaSearch( start-state, actions, goal-test )
                                        returns action-sequence or failure
    static: agenda, a node sequence, initially contains one node (for start-state)
    loop do
        if Empty(agenda) then return failure
        if goal-test(First(agenda)) then return GetActions(First(agenda))
        agenda ← Queuing-Fn(Rest(agenda), Expand(First(agenda), actions))
    end loop
```

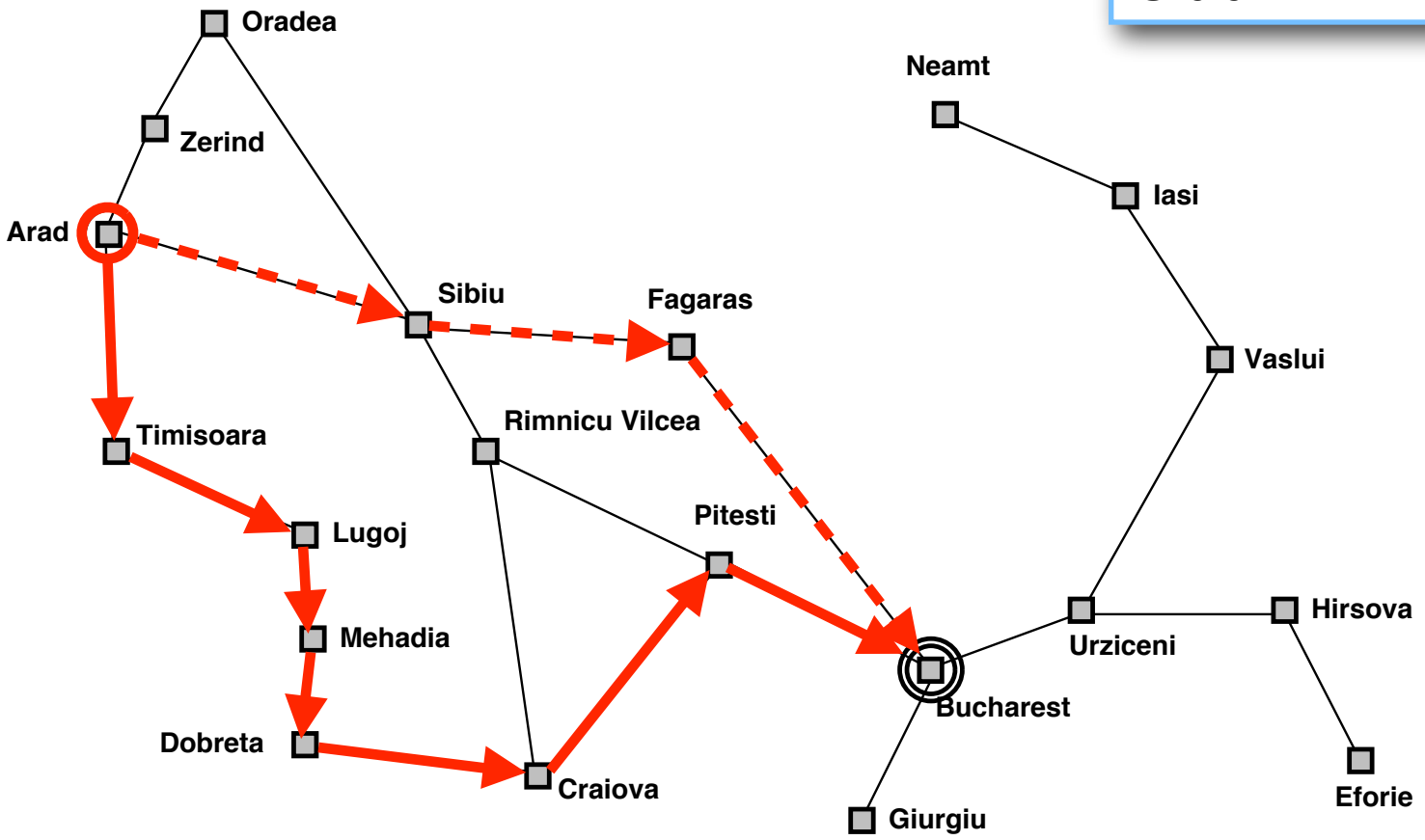
- The Queuing-Fn function determines what kind of search we are doing
- The Expand function produces a set of nodes representing states achieved by applying all possible actions to the given state
- This is generally called Agenda-based search

Search & the Queuing-Fn

Queen Mary University of London

- Searching this way takes us to Bucharest in 3 steps
  - but only because of the order we picked the cities in





if we'd chosen rightmost order, we would have gone via Timisoara, Dobreta and Pitesti – wasting time and fuel

Search & the Queuing-Fn

Queen Mary University of London

- This is an example of Depth-First Search (DFS)
  - we construct the search tree by going *down* it, only going *across* it when going down fails
  - depth-first search can fail when the tree is infinitely deep
    - the space contains cycles or is generated from continuous values
  - depth-first search depends (crucially) on the order in which operators are applied/nodes are expanded
    - it may not find things that are present
    - it may find long action sequences before short ones (usually, that's unhelpful)
- In terms of agendas, we implement DFS with a Queuing-Fn that
  - removes the current node from the front of the agenda
  - appends its children to the *front* of the agenda
  - corresponds to a *stack* (LIFO) data structure

Search & the Queuing-Fn

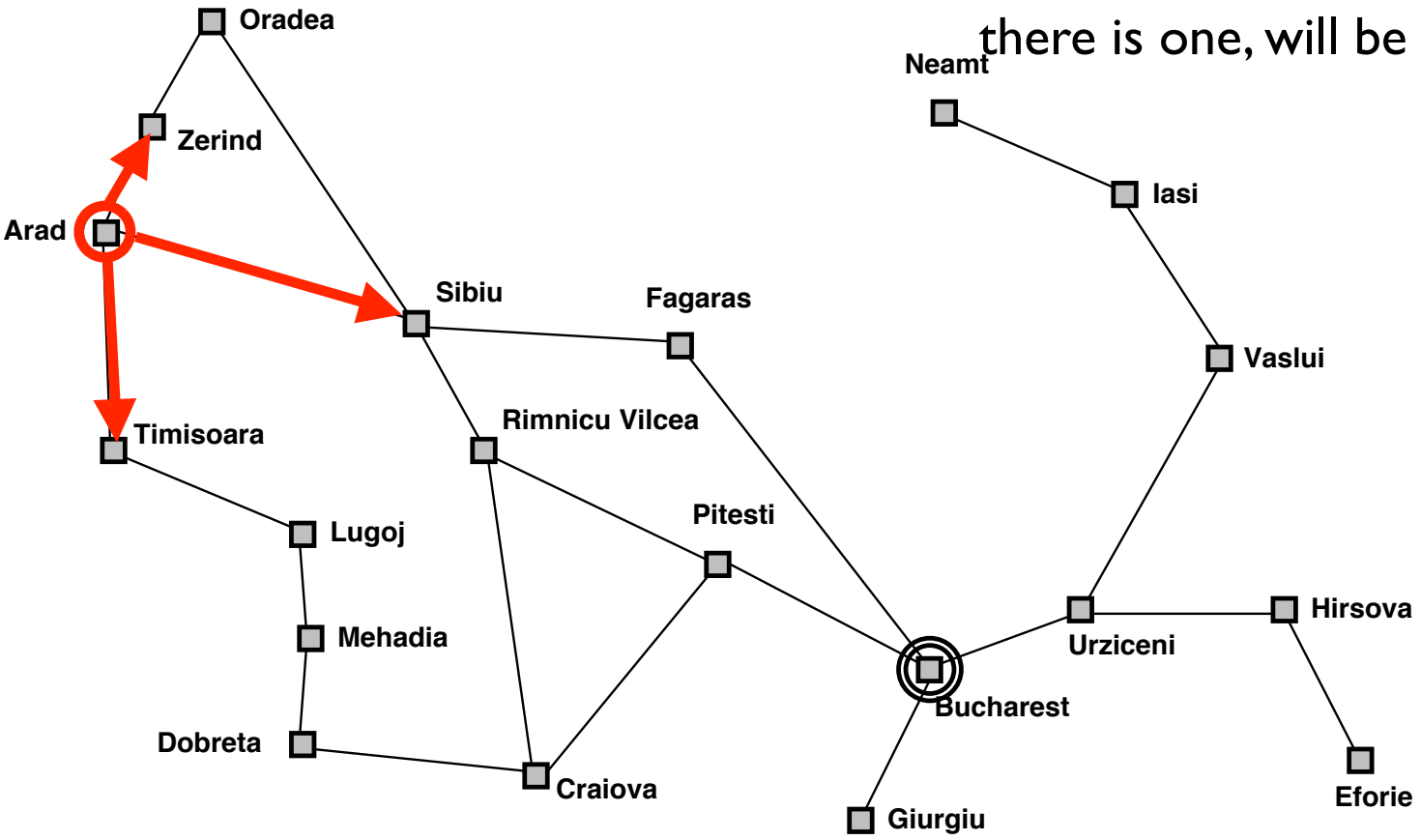
Queen Mary University of London

- An alternative is Breadth-First Search (BFS)
  - we construct the search tree by going *across* it, repeatedly, replacing each node with its children
  - breadth-first search guarantees to find the solution if there is one
  - breadth-first search guarantees to find the shortest action sequence
  - but it causes a combinatorial explosion of nodes in the agenda, and so is unsuitable for very large search spaces
    - most real AI problems have very large search spaces
- In terms of agendas, we implement BFS with a Queuing-Fn that
  - removes the current node from the front of the agenda
  - appends its children to the *back* of the agenda
  - corresponds to a *queue* (FIFO) data structure

Breadth-first search in Romania

Queen Mary University of London

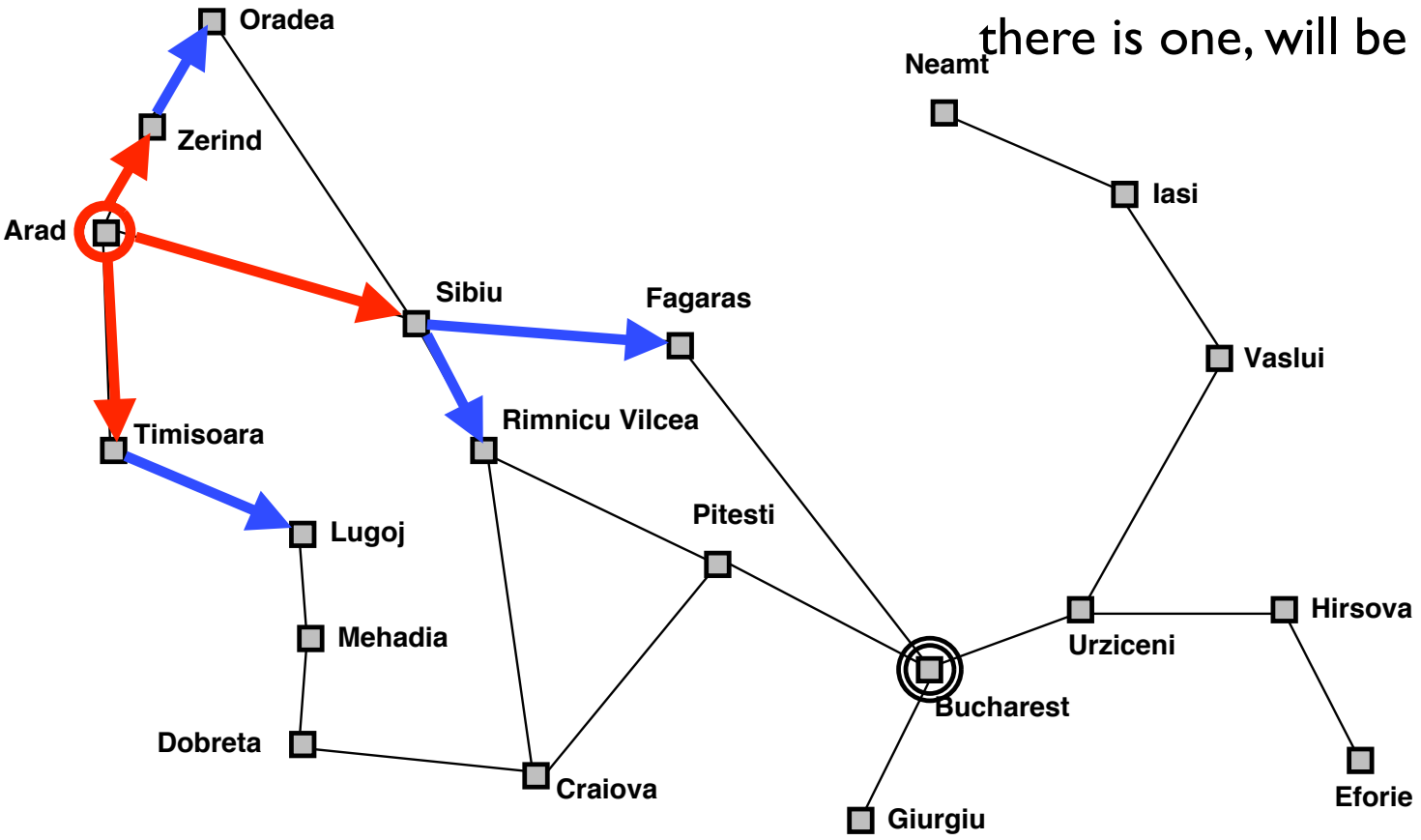
- Here the different depths are given different colours
  - we can see the different layers of the tree as the search proceeds
  - and it's clear that the shortest solution, if there is one, will be found



Breadth-first search in Romania

Queen Mary University of London

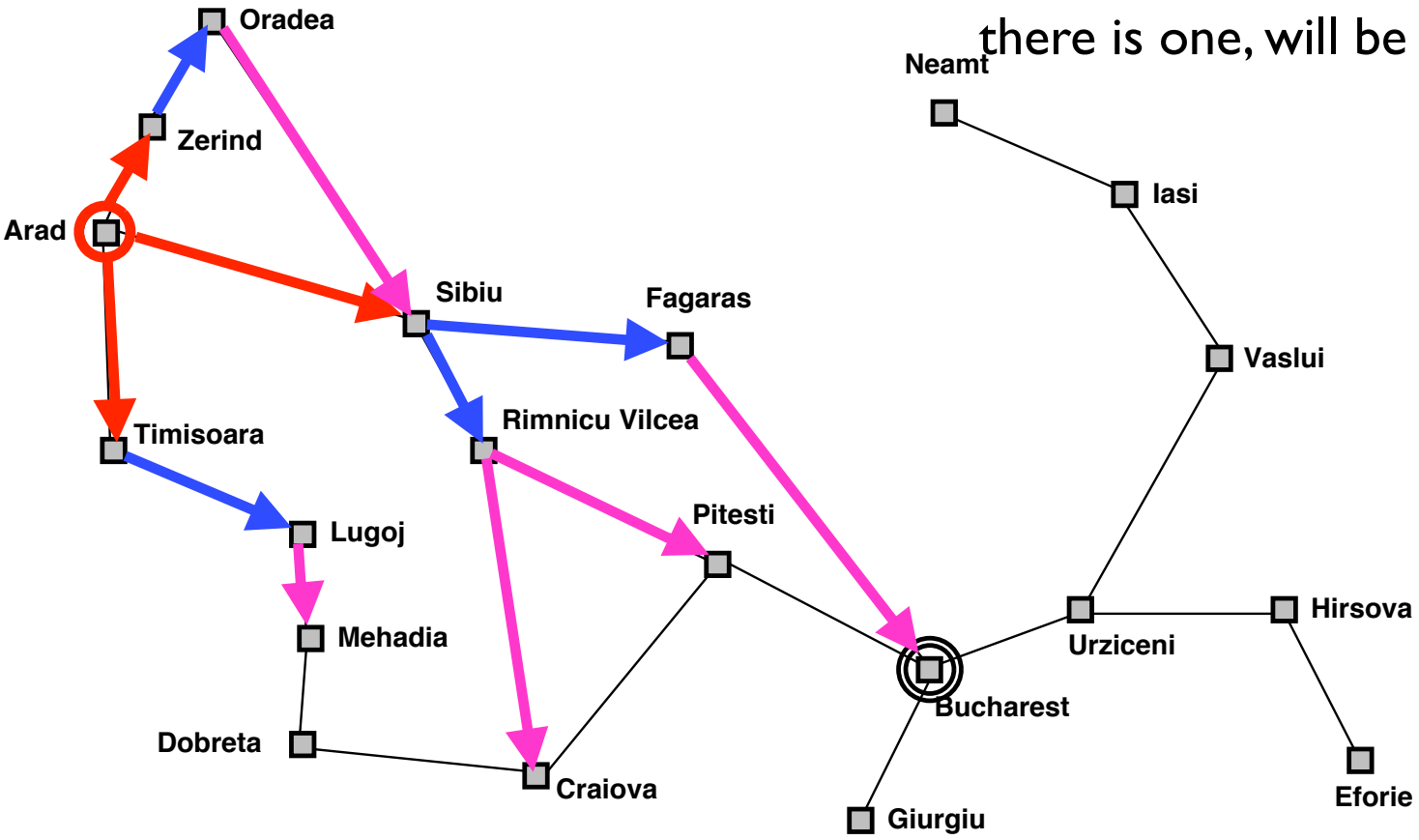
- Here the different depths are given different colours
  - we can see the different layers of the tree as the search proceeds
  - and it's clear that the shortest solution, if there is one, will be found



Breadth-first search in Romania

Queen Mary University of London

- Here the different depths are given different colours
  - we can see the different layers of the tree as the search proceeds
  - and it's clear that the shortest solution, if there is one, will be found



Measuring search effectiveness

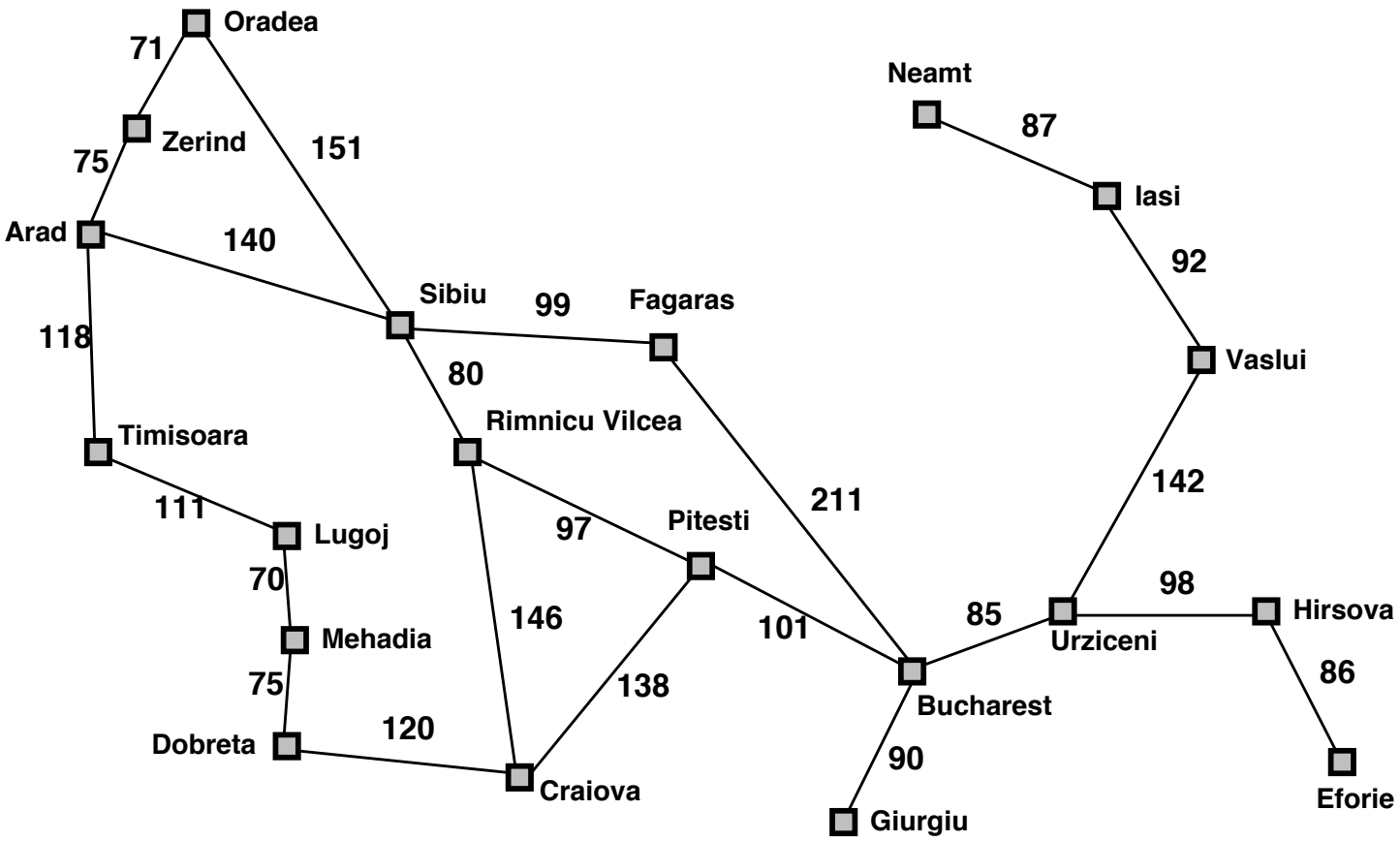
Queen Mary University of London

- To determine the utility of a particular path, we need
  - cost: a function that assigns a cost to a path (to differentiate between solutions)
- To evaluate a search strategy or algorithm, need to ask
  - does it find a solution?
  - is it a good solution (one with a low path cost)?
  - is it the best solution (optimal)?
  - how much time and memory are required to find a solution?
- Effectiveness of search depends on the objectives. Compare
  - searching for a route to Bucharest (many routes may be satisfactory)
  - searching for the next prime number (there's only one correct solution)

Measuring search effectiveness

Queen Mary University of London

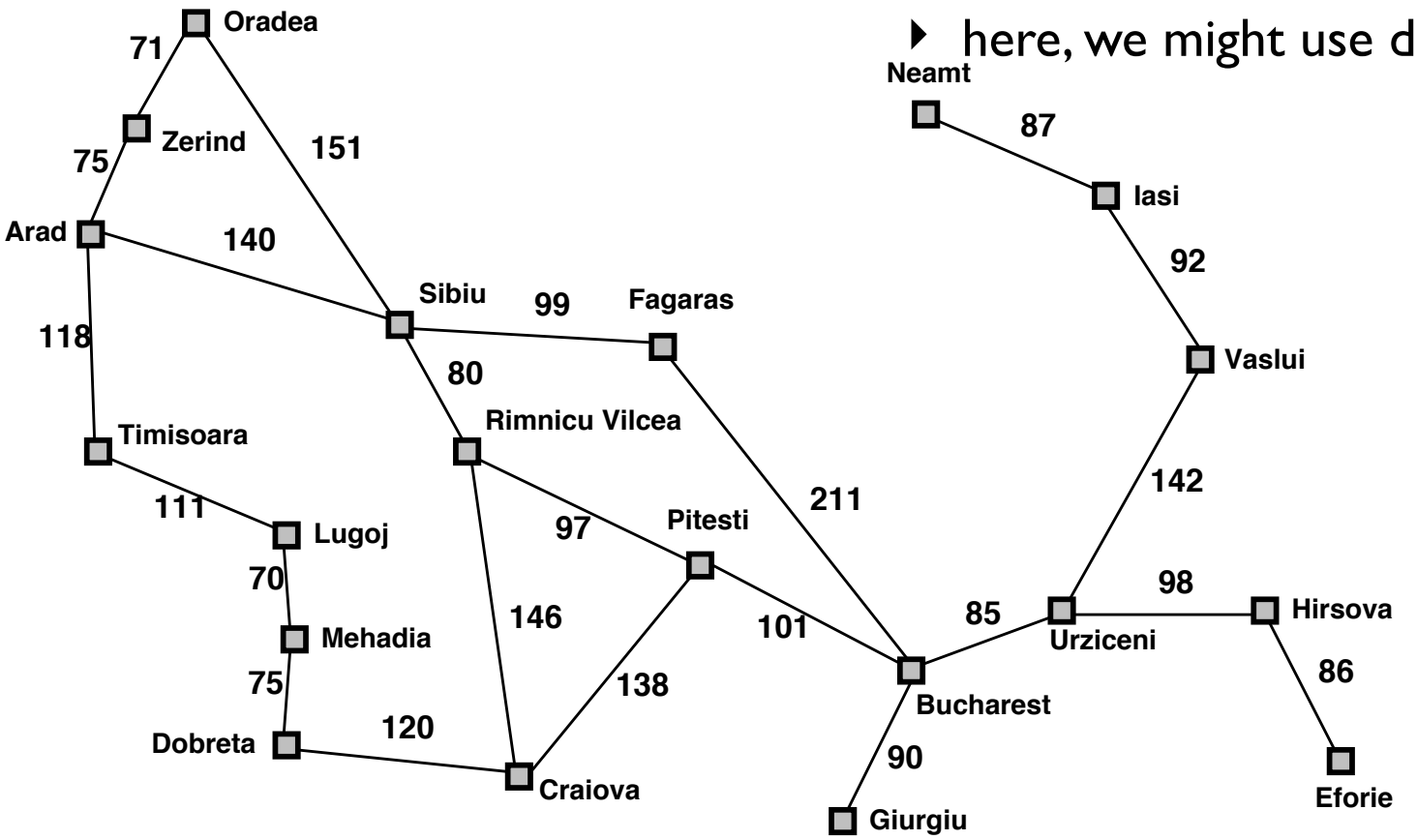
- Now consider the efficiency of the planned route
  - going via Sibiu and Fagaras has the fewest towns
  - but going via Sibiu and Pitesti is shorter



Measuring search effectiveness

Queen Mary University of London

- We can compute the *utility* of a route by a function which implements the performance measure, in terms of the problem formulation
  - here, we might use distance for each road



Search & the Queuing-Fn

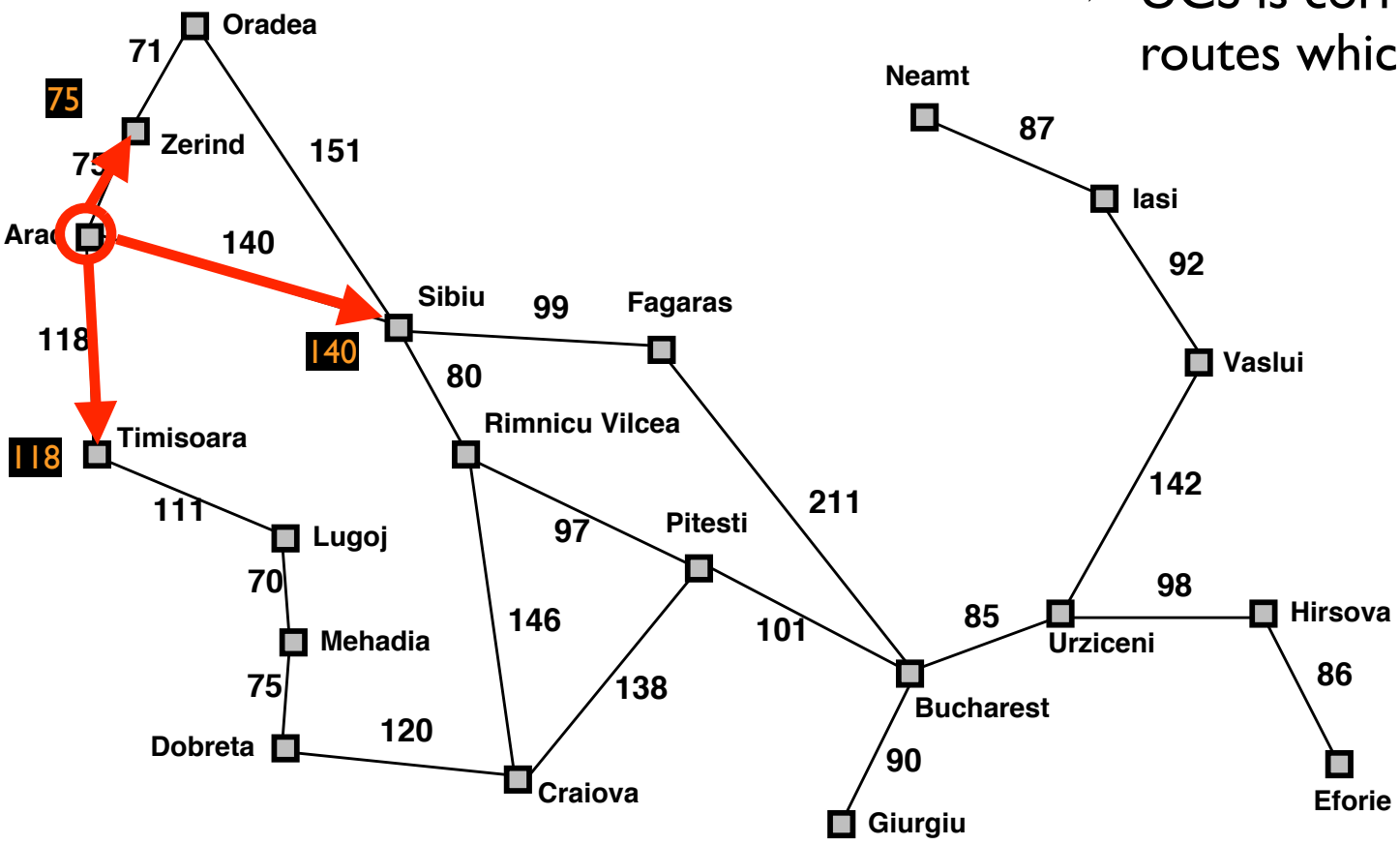
Queen Mary University of London

- We can implement this using Uniform-Cost Search (UCS)
  - Also called Best-First Search (Note: acronym "BFS" is already taken - *Breadth*-First Search)
  - we construct the search tree by picking the *best* node in the agenda after each expansion, according to the utility function, expanding it, and so on
  - uniform-cost search guarantees to find the solution if there is one
  - uniform-cost search guarantees to find the best action sequence in terms of *utility*
  - but it cannot guarantee to use less memory than BFS
    - most real AI problems have very large search spaces
- In terms of agendas, implement UCS with a Queuing-Fn that
  - removes the current node from the agenda
  - adds its children to the agenda & sorts the *whole agenda* by the utility function
    - and stores the cost-so-far at each node in the agenda, to avoid recalculation
  - corresponds to a *priority queue* data structure

Uniform cost search in Romania

Queen Mary University of London

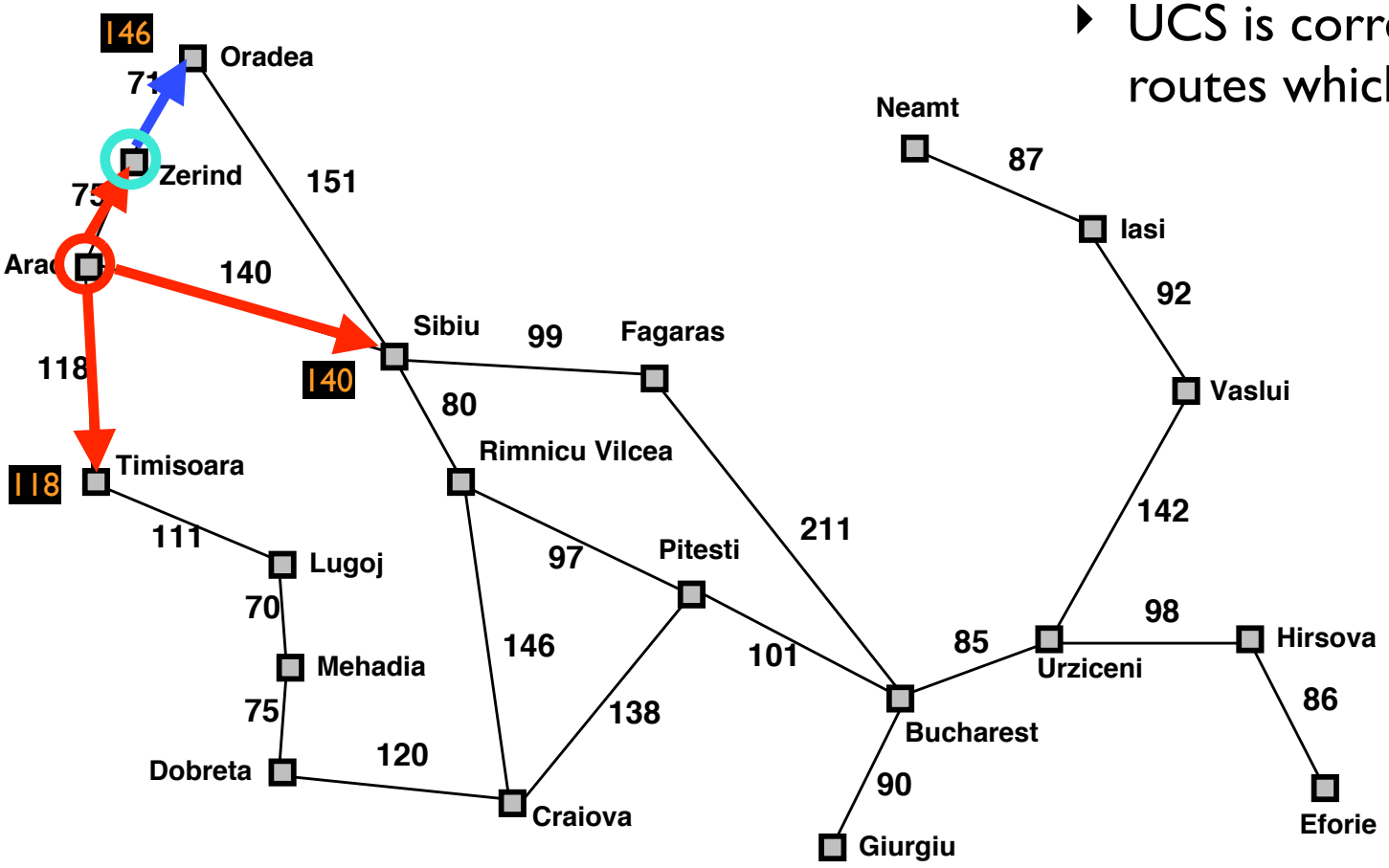
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

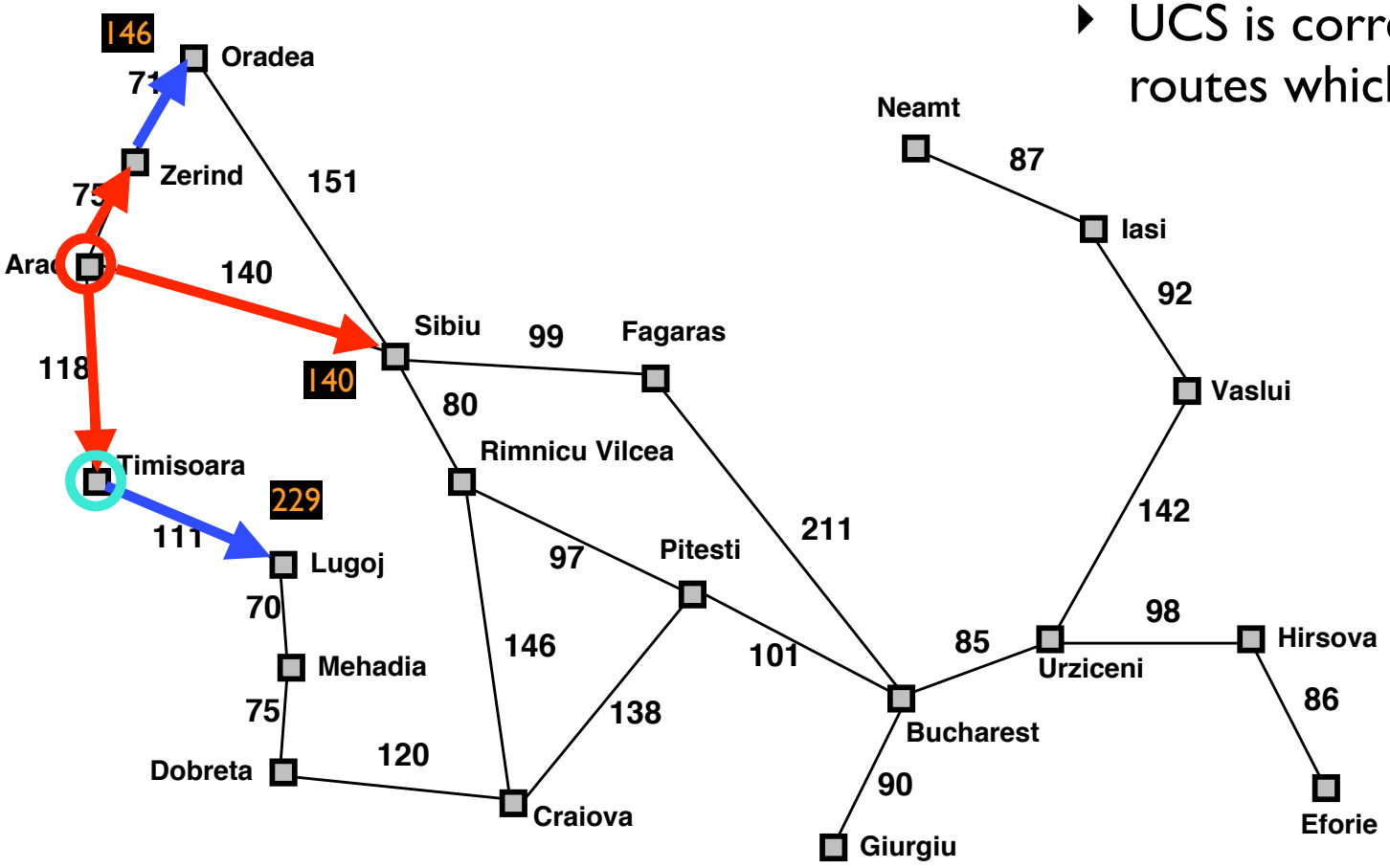
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

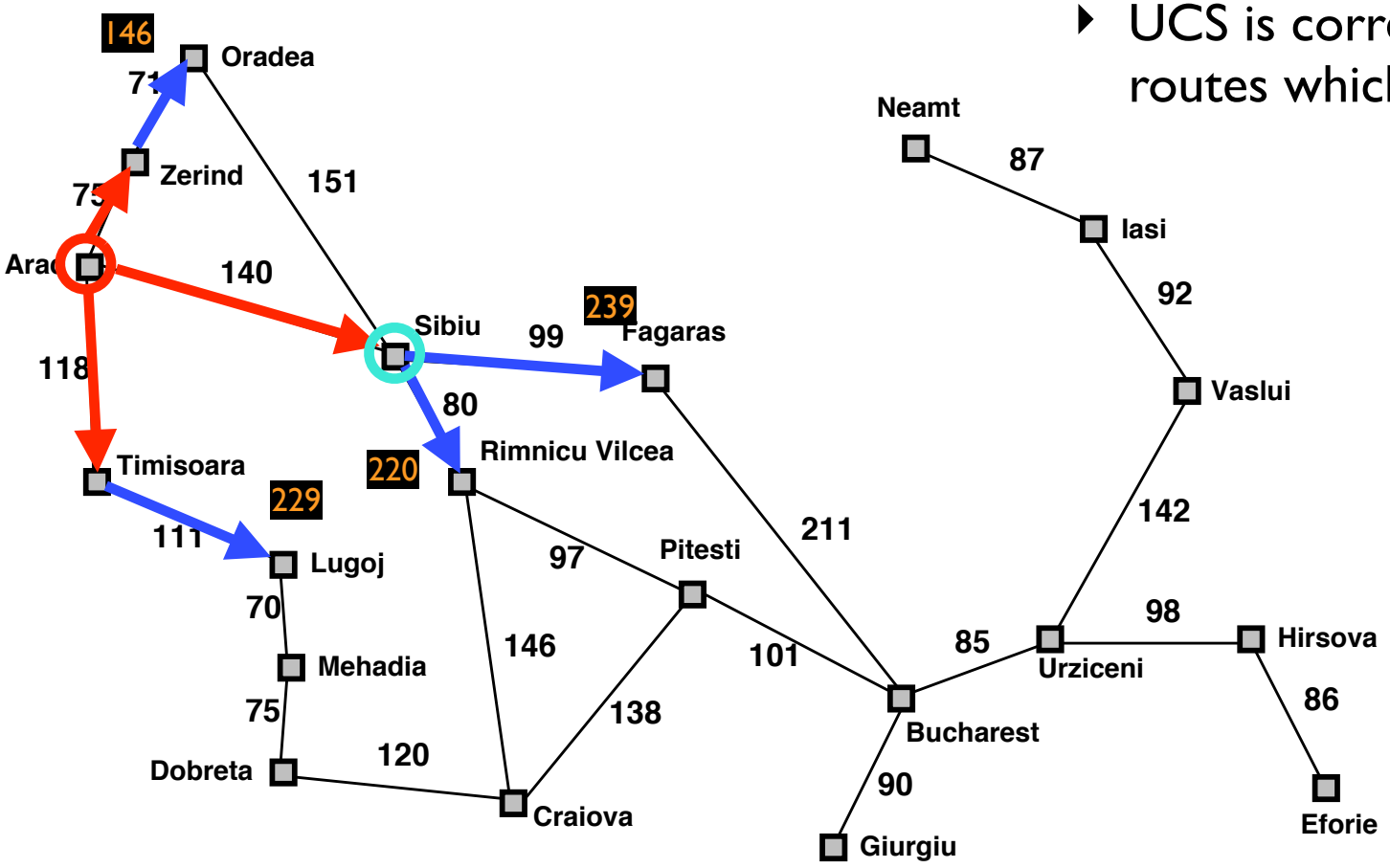
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

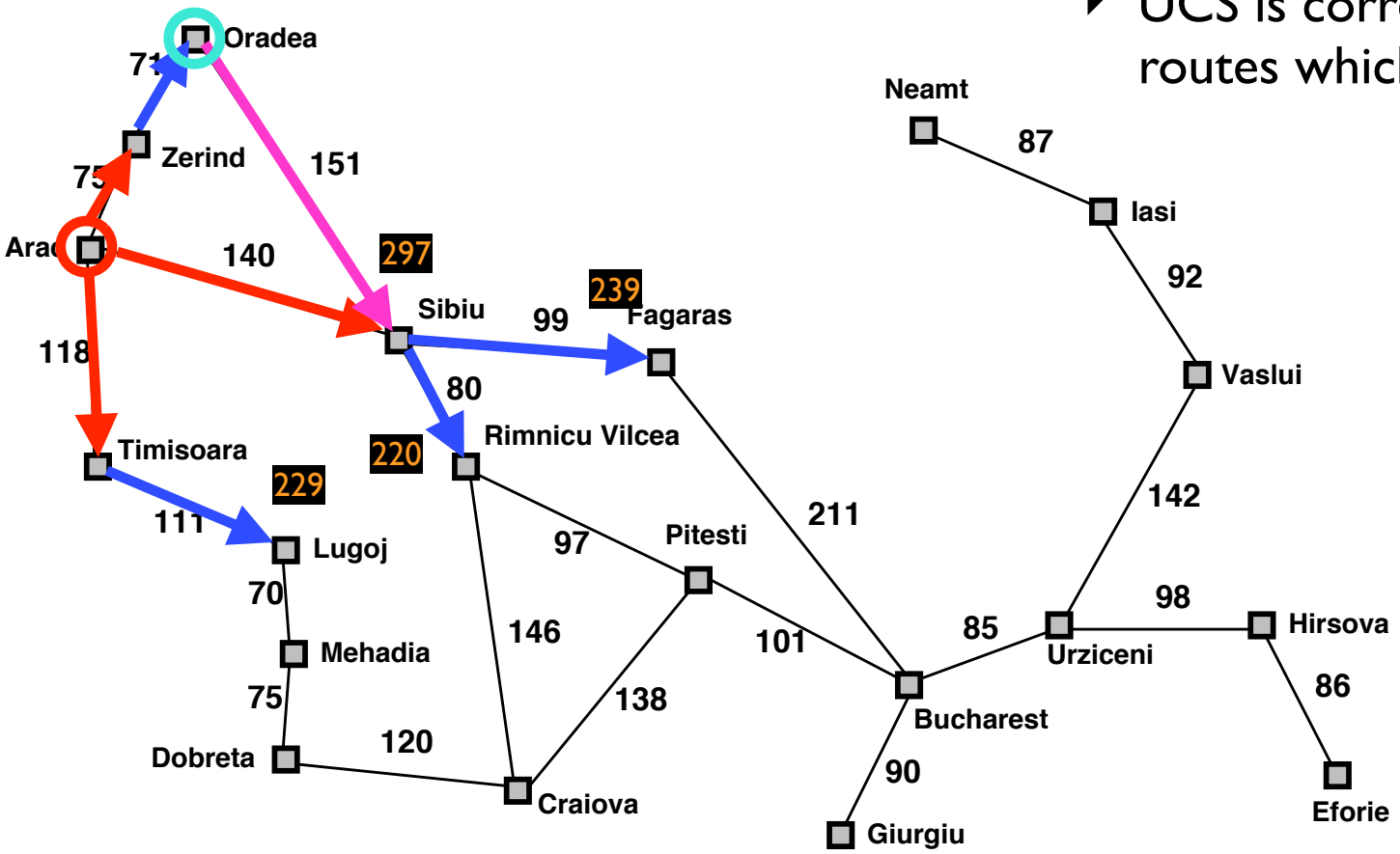
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

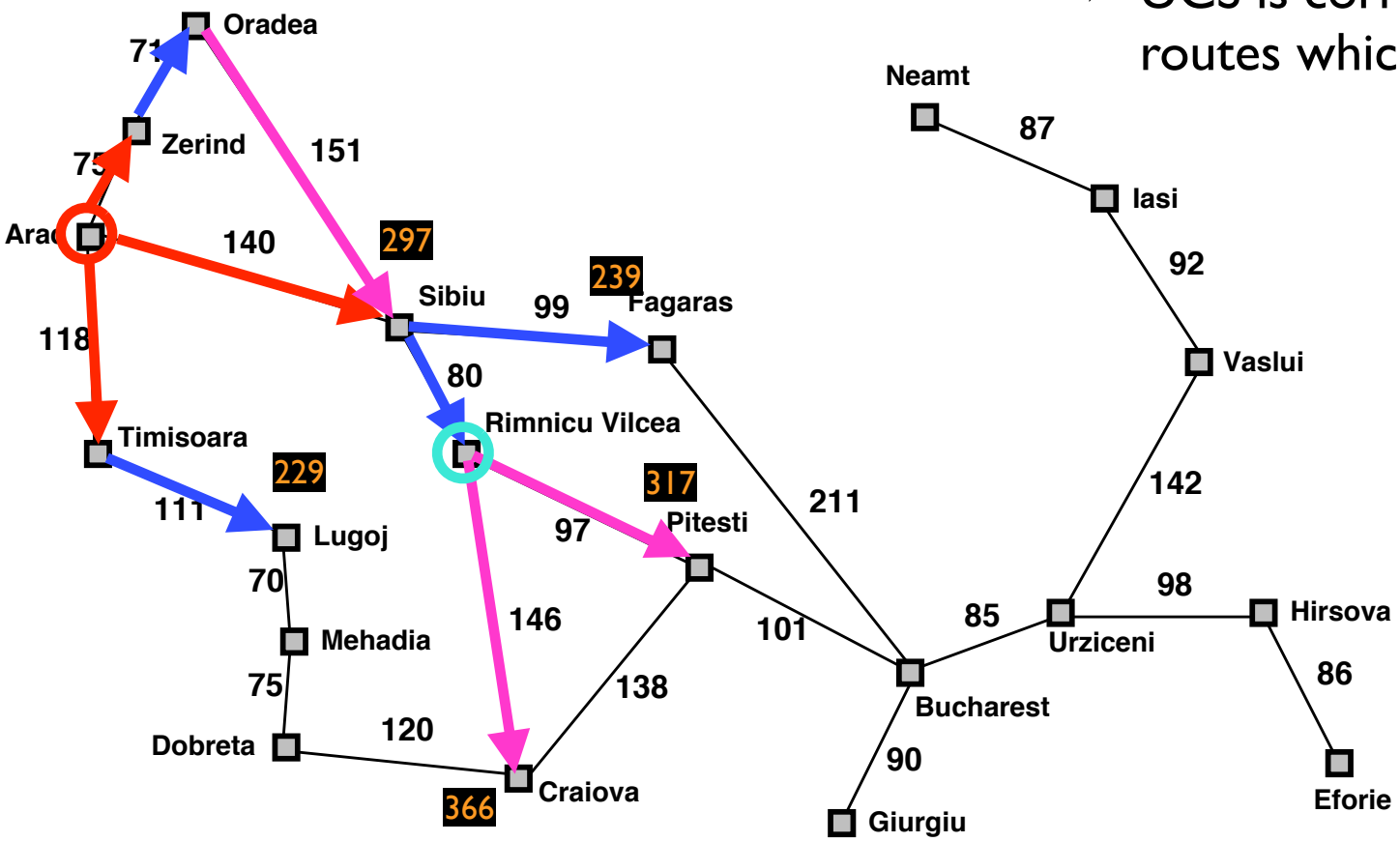
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

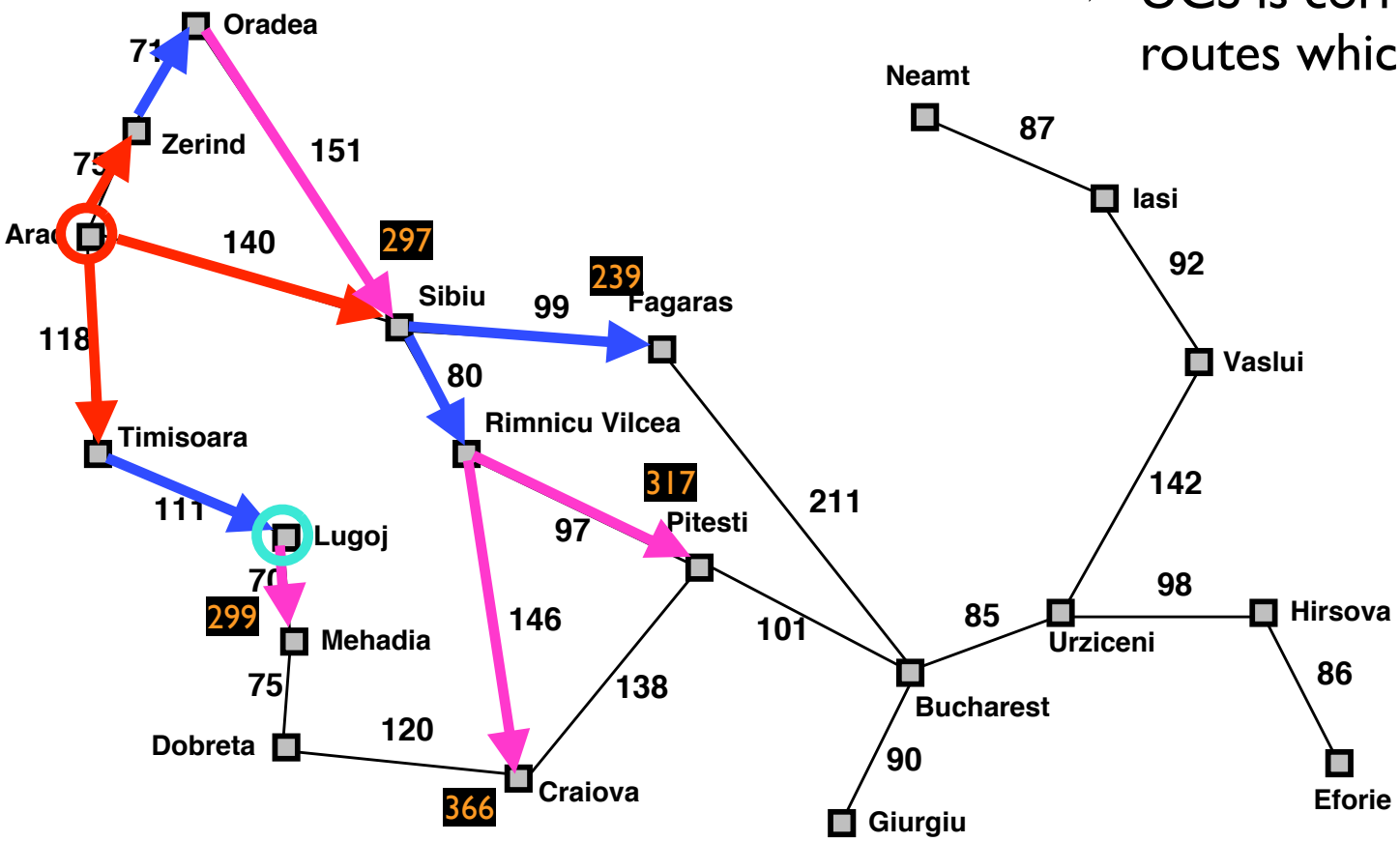
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

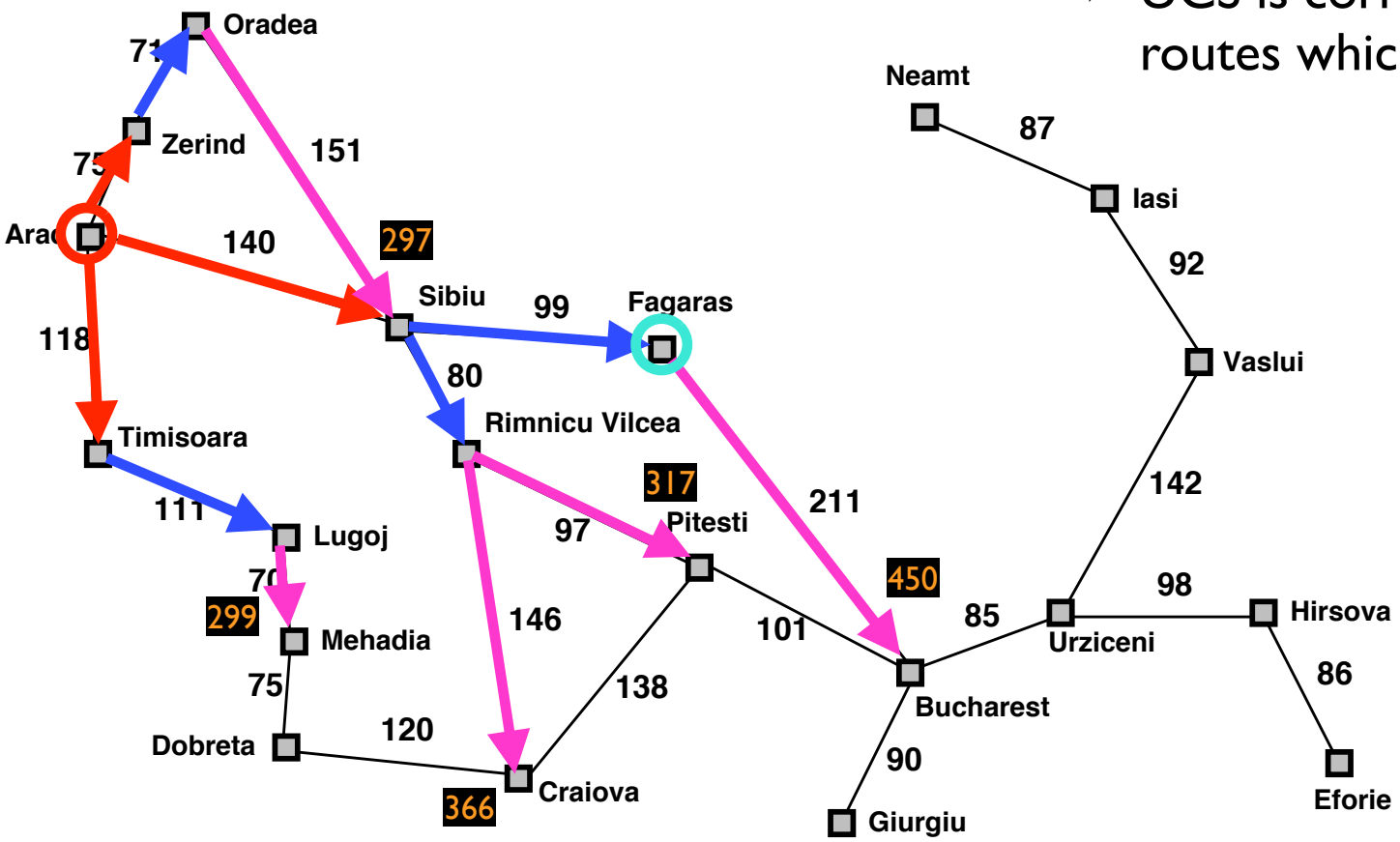
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

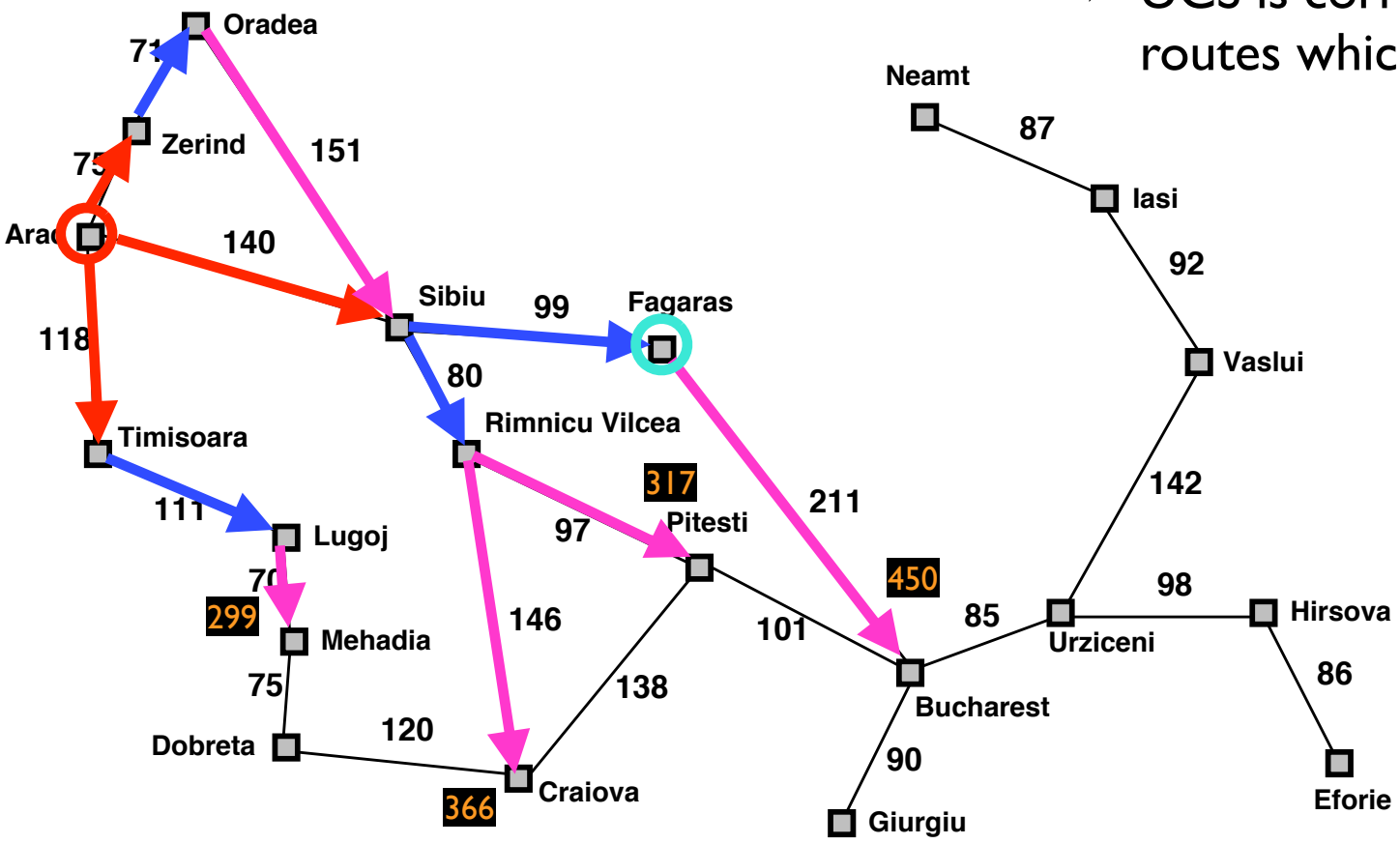
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

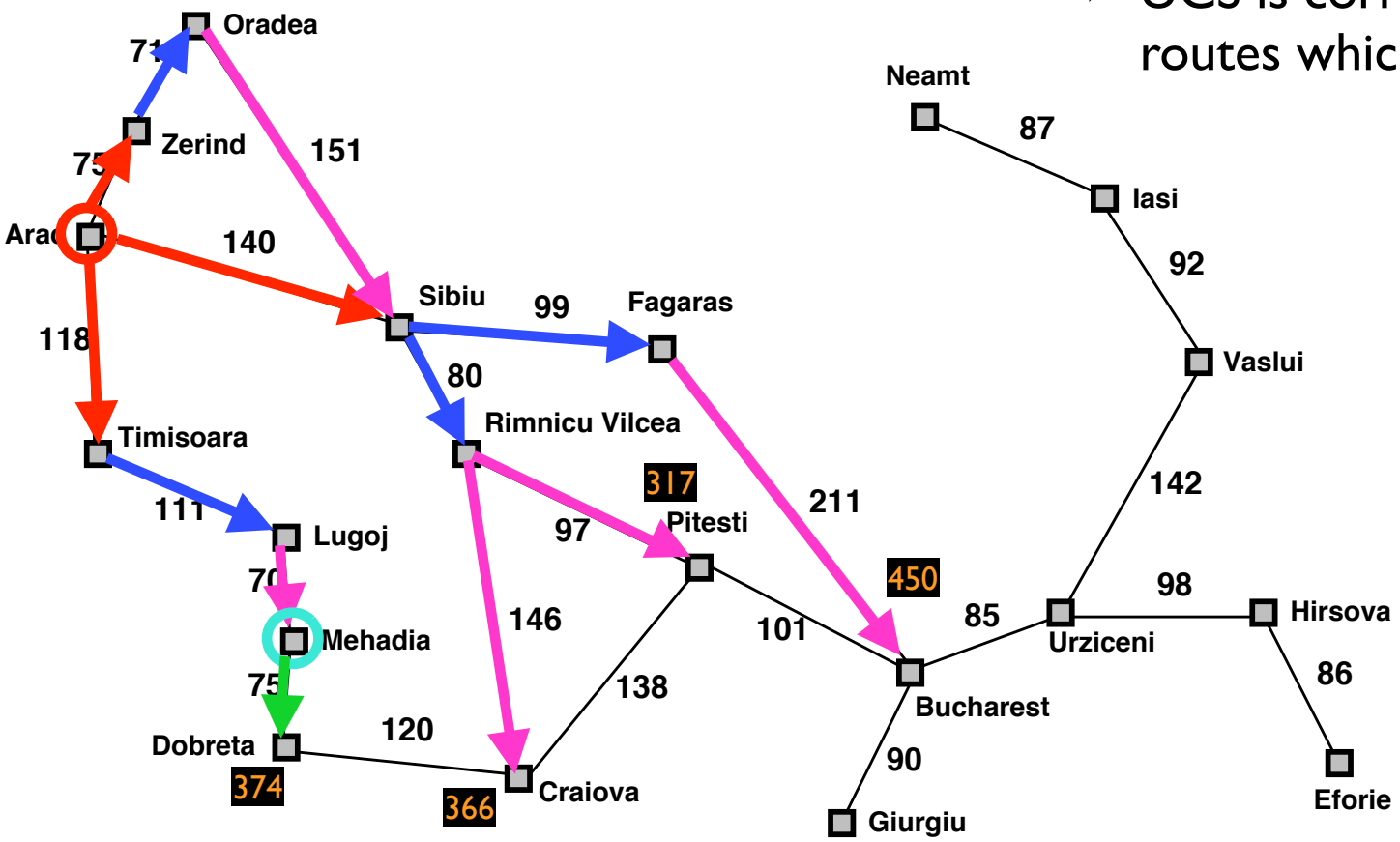
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

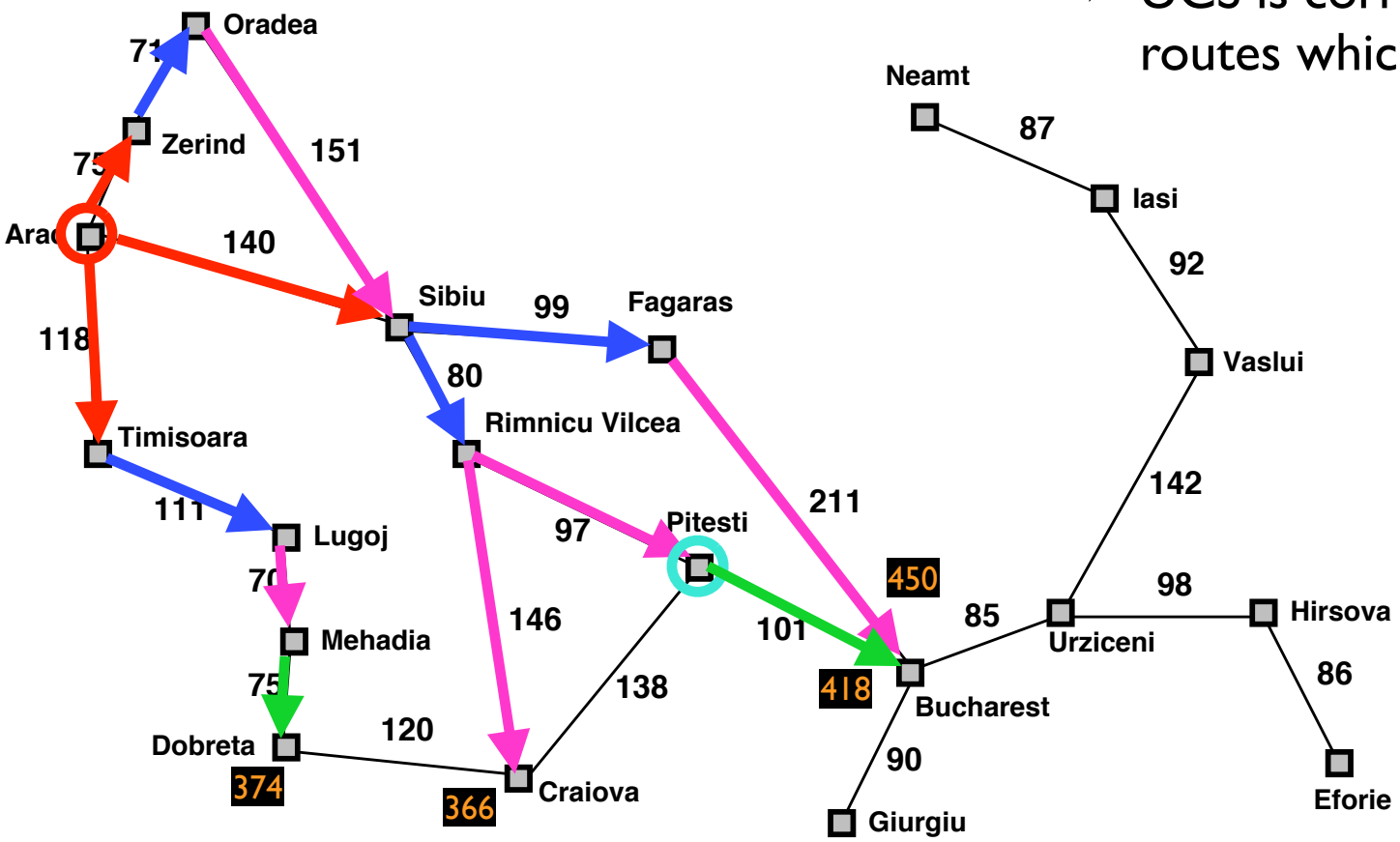
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

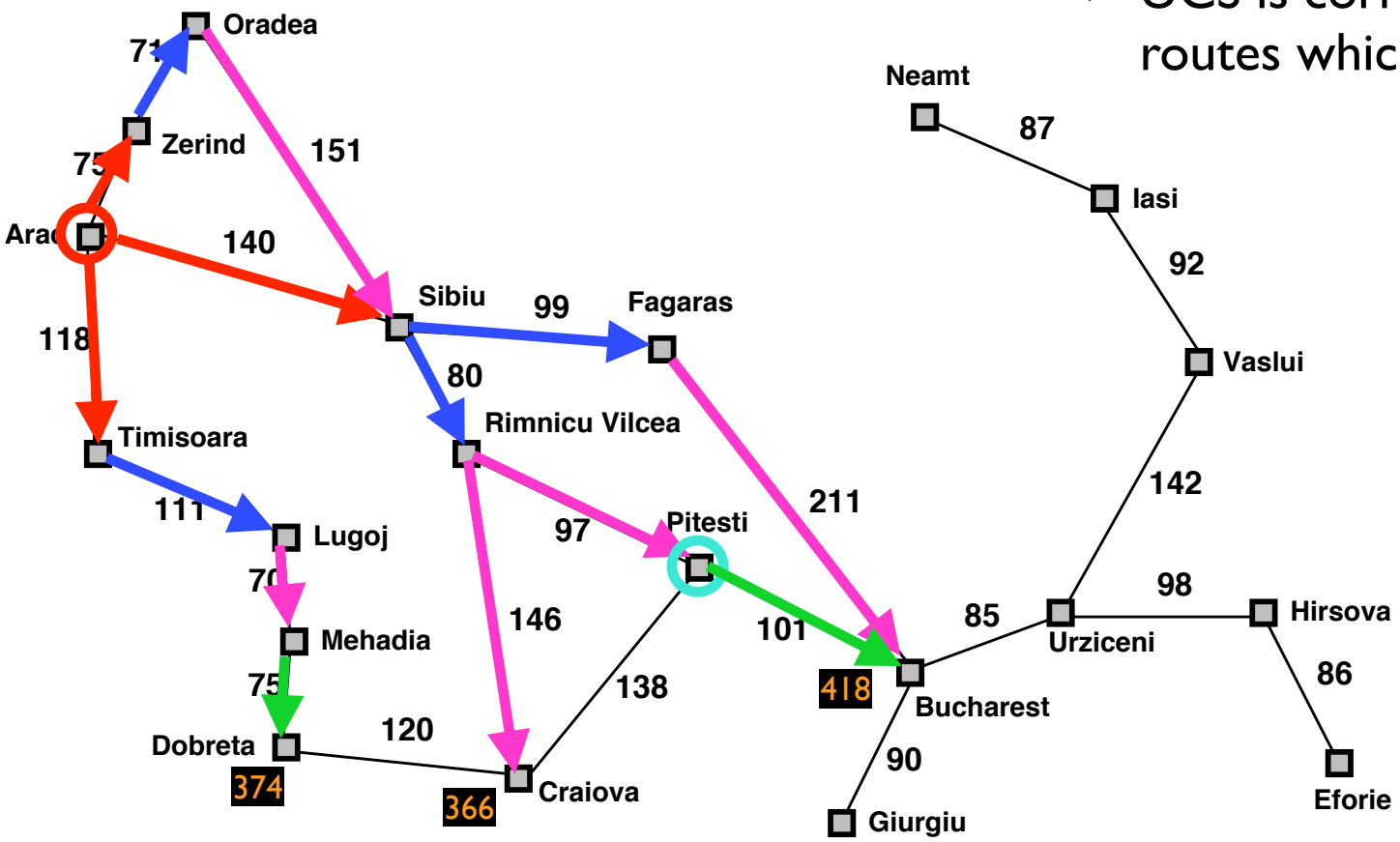
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

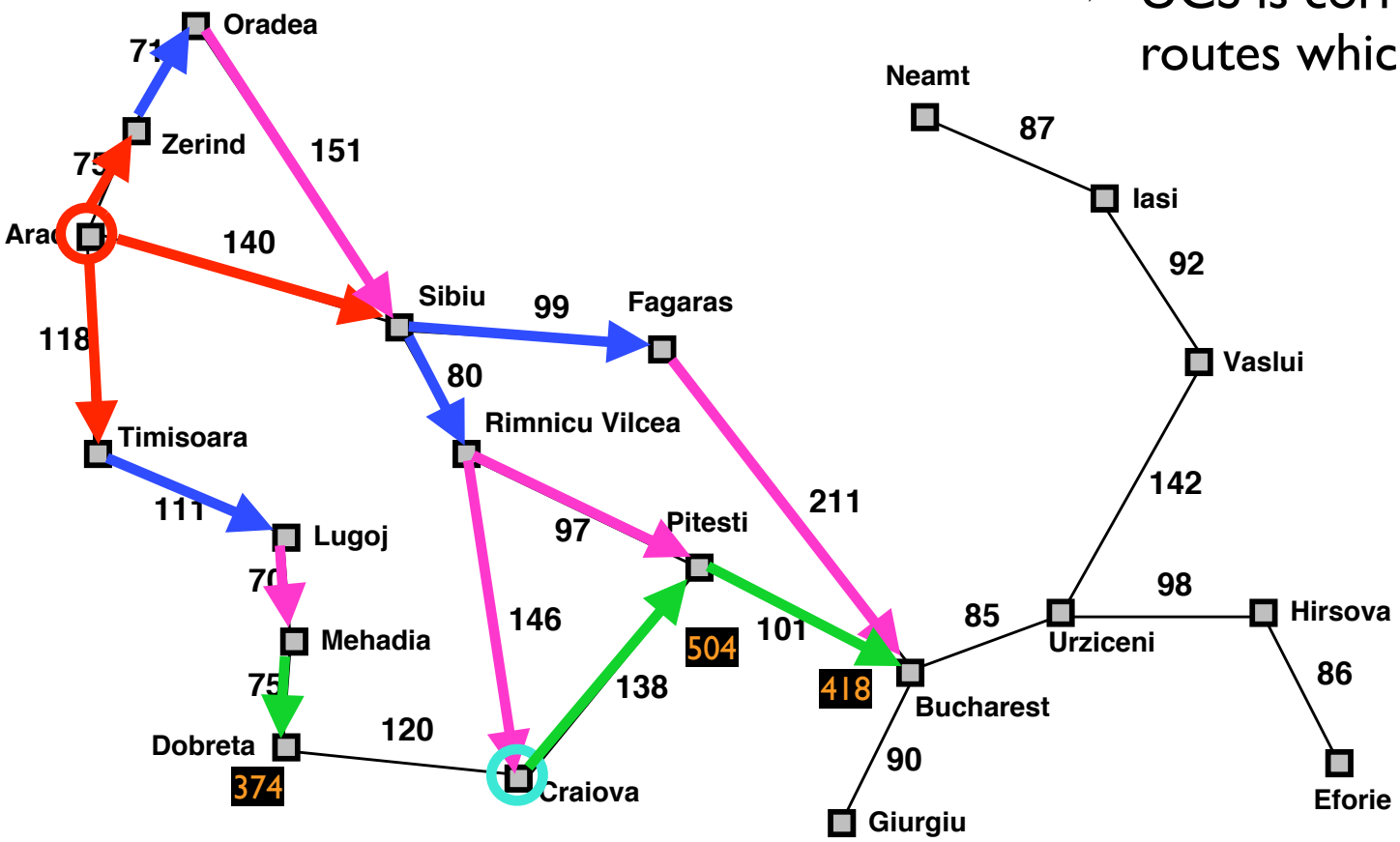
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

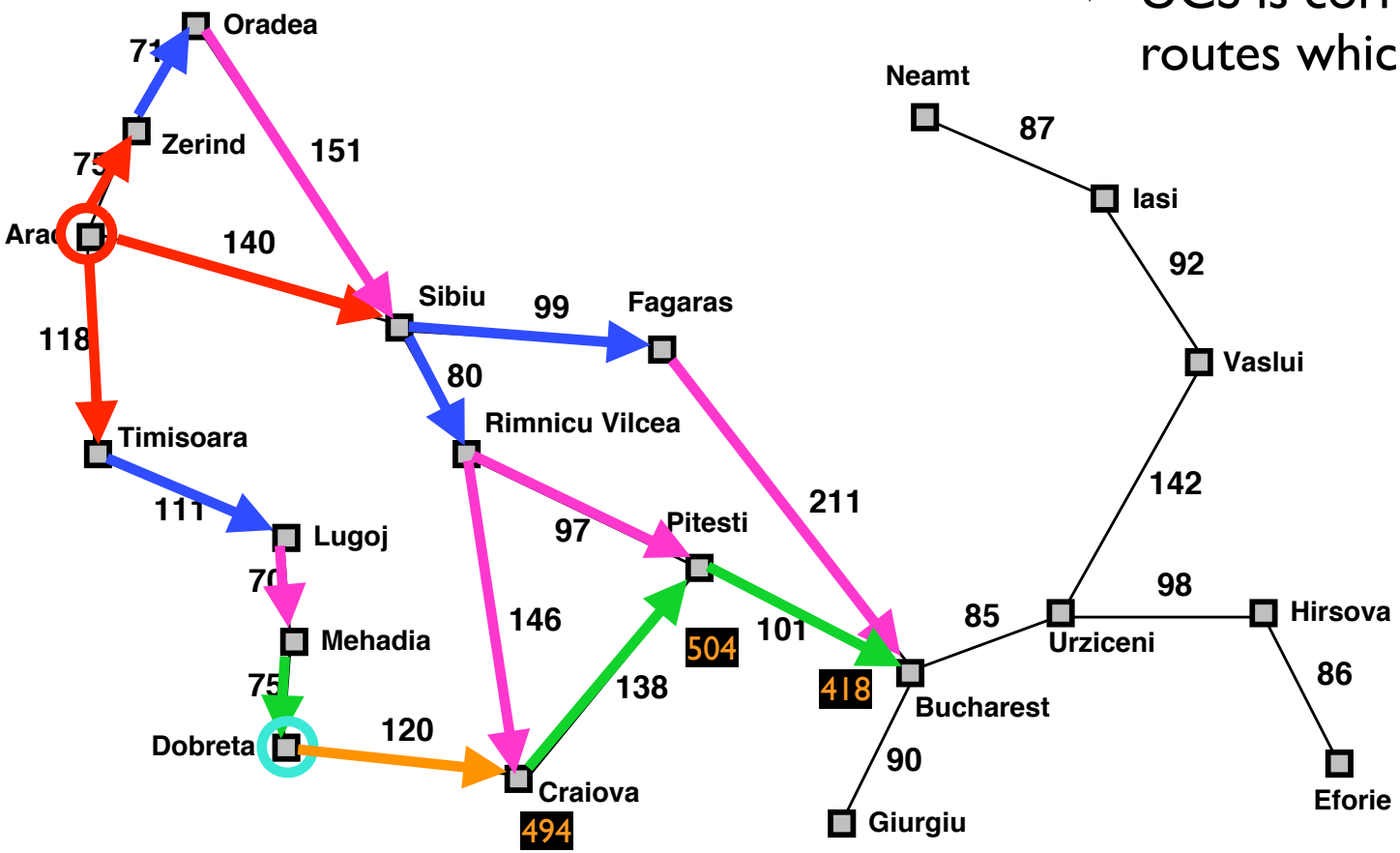
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

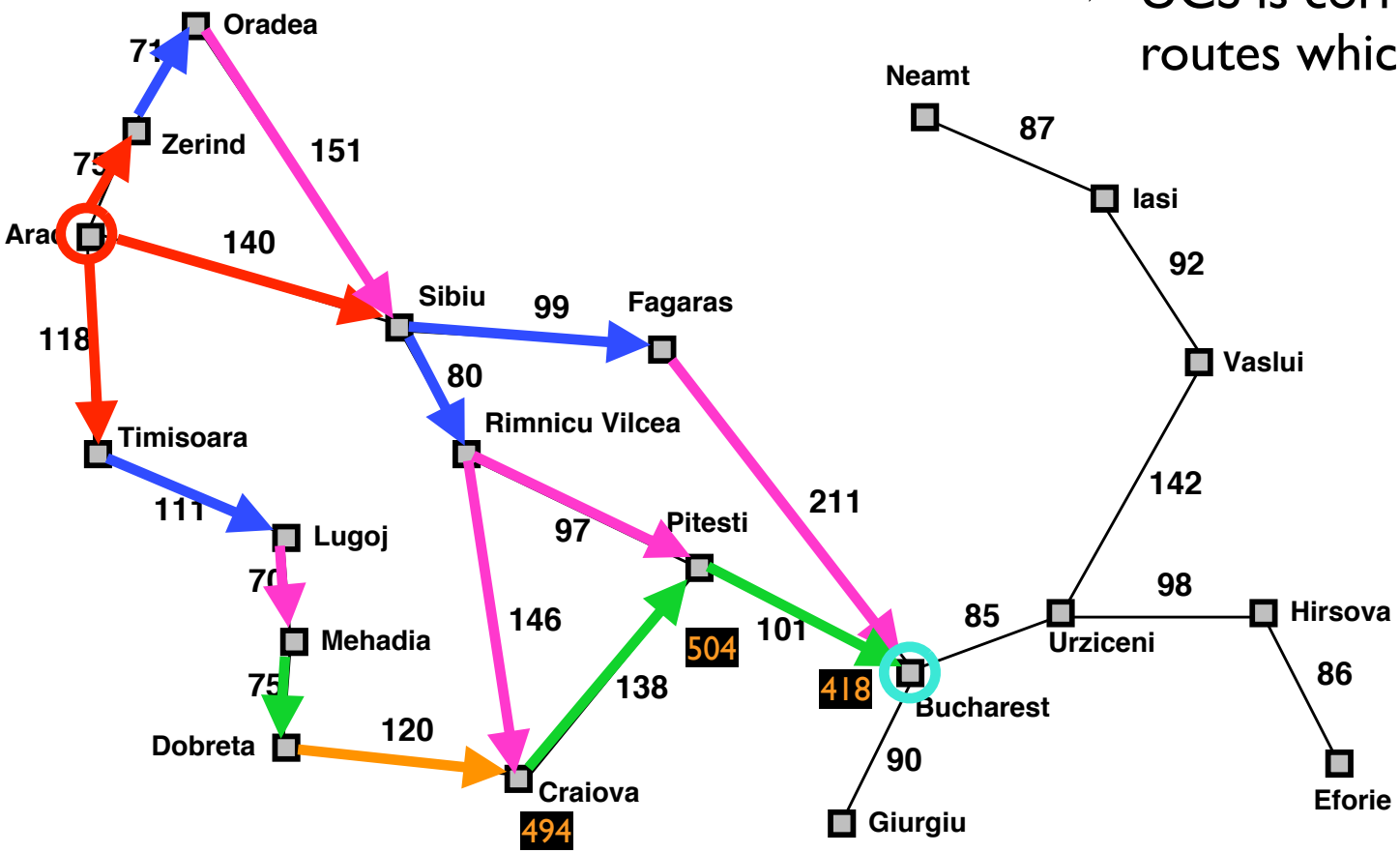
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Uniform cost search in Romania

Queen Mary University of London

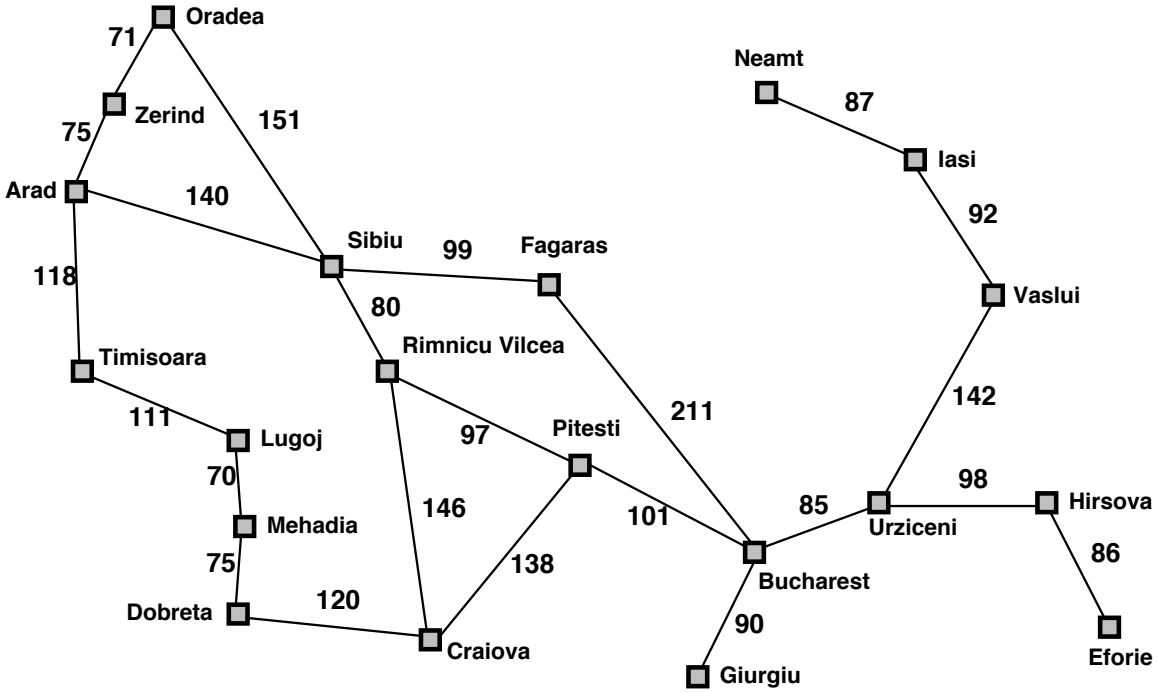
- Using UCS to consider real distances
  - Here BFS would get us a *wrong* answer
  - UCS is correct, but spends time searchi routes which are "obviously" wrong



Formulating the problem

Queen Mary University of London

- Now let's look at the full specification of this problem
  - Initial state: in Arad
  - Operator: drive between two cities
  - State space: cities (20)
  - Goal test: in Bucharest?
- Path cost: one of
  - total mileage
  - total expected travel time
  - step count in search
- Many details are left out
  - *abstraction*



Analysing search spaces

Queen Mary University of London

- A strategy is defined by its order of node expansion
- *Uninformed* strategies use *only information in the problem formulation*
  - Sometimes called *blind* search or *brute-force* search
    - Breadth-first search
    - Depth-first search
    - Uniform-cost search (this "feels" informed, but isn't, by this definition)
- *Informed* or *heuristic* search strategies use a quality measure not strictly in the problem formulation to guide the order of node expansion
  - Estimates how promising a node is
  - e.g., An informed agent might notice that the goal state, Bucharest, is southeast of Arad, and that only Sibiu is in that direction, so it is likely to be the best choice
    - Algorithm A and Algorithm A*

Analysing search spaces for cost

Queen Mary University of London

- A strategy is evaluated along the following dimensions
  - *completeness* does it always find a solution if one exists?
  - *time complexity* number of steps e.g. nodes generated
  - *space complexity* e.g. maximum number of nodes in memory at once
  - *optimality* does it always find a least-cost solution?
- Time and space complexity are measured in terms of the search tree
  - b maximum branching factor of the search tree ("bushiness" of tree)
  - d depth of the least-cost solution
  - m maximum depth of the state space (may be ∞)
- Even the best strategies need to avoid repeated nodes
  - trade-off between time and space (see Russell and Norvig)

Analysing search algorithms for cost

- So far, we've looked at Depth First, Breadth First, and Uniform Cost Search in practice
  - now, the theory

Analysing search algorithms for cost

Queen Mary University of London

- Depth first search (DFS)
  - Complete? No, DFS fails in infinite depth search spaces and spaces with loops
  - Time? $O(b^m)$: terrible if m is much larger than d, but if search tree is "bushy", may be much faster than breadth first
  - Space? $O(bm)$, i.e., space linear in length of action sequence! Need only store a single path from the root to the leaf node, along with remaining unexpanded sibling nodes for each node on path
  - Optimal? No
  - Pros: For problems with many solutions, DFS may be faster than BFS because it has a good chance of finding a solution after exploring only a small portion of the whole space
  - Cons: can get stuck down a wrong search path

Analysing search algorithms for cost

Queen Mary University of London

- Breadth first search (BFS)
  - Complete? Yes, all nodes are examined if b is finite
  - Time? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exponential in b
  - Space? $O(b^d)$ – keeps every node in memory
  - Optimal? Only if cost = 1 per step, not in general
    - If there are several solutions it will find the *shallowest* goal first
  - Drawbacks: Memory requirement to store nodes is a big problem
    - Assuming b = 10, expansion of 1000 nodes/sec, and 100 bytes/node:

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	0.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Analysing search algorithms for cost

Queen Mary University of London

- Uniform-Cost Search (UCS)
  - Shares dynamic properties of DFS and BFS
  - Requires step cost information in problem formation and path cost in agenda
  - Time? Number of nodes with path cost less than optimal solution
  - Space? Number of nodes with path cost less than optimal solution
  - Optimal? Yes – finds the cheapest solution as long as step cost > 0
  - First solution reported (but not necessarily encountered) is always the cheapest
    - does not explore the whole tree and avoids looping
    - but can visit the same state multiple times if there is more than one route through a node

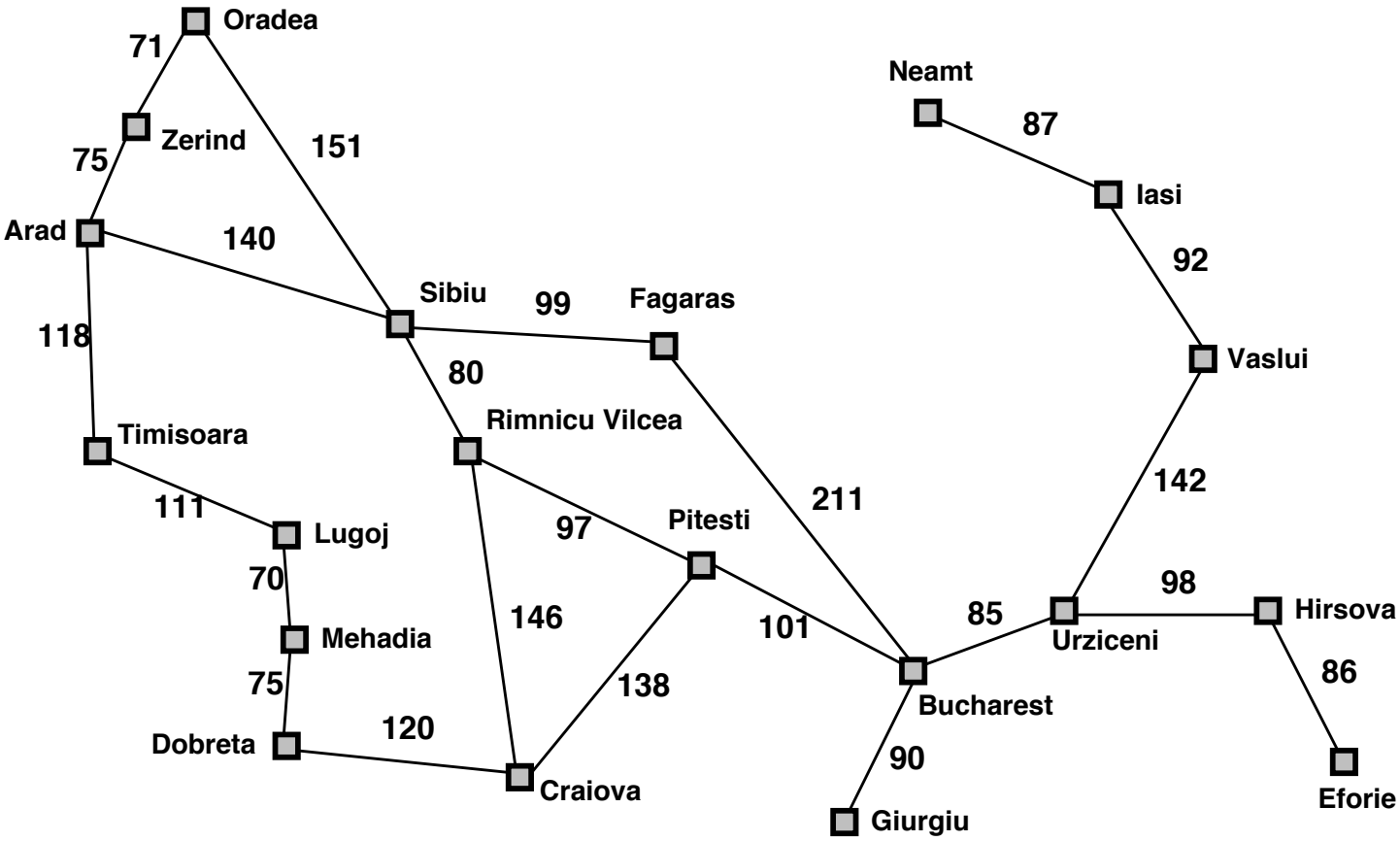
Depth-bounded search: IDS

Queen Mary University of London

- *Iterative-deepening* search is a compromise between DFS and BFS
  - Do DFS, but include a depth bound, forcing BFS-like behaviour
  - In the agenda-based algorithm, we keep track of the depth of each node in the agenda, then
    - Start with depth-cutoff = 1
    - Search, expanding only nodes with depth < depth-cutoff
    - if unsuccessful, increment depth-cutoff and repeat

Exercise: IDS on holiday

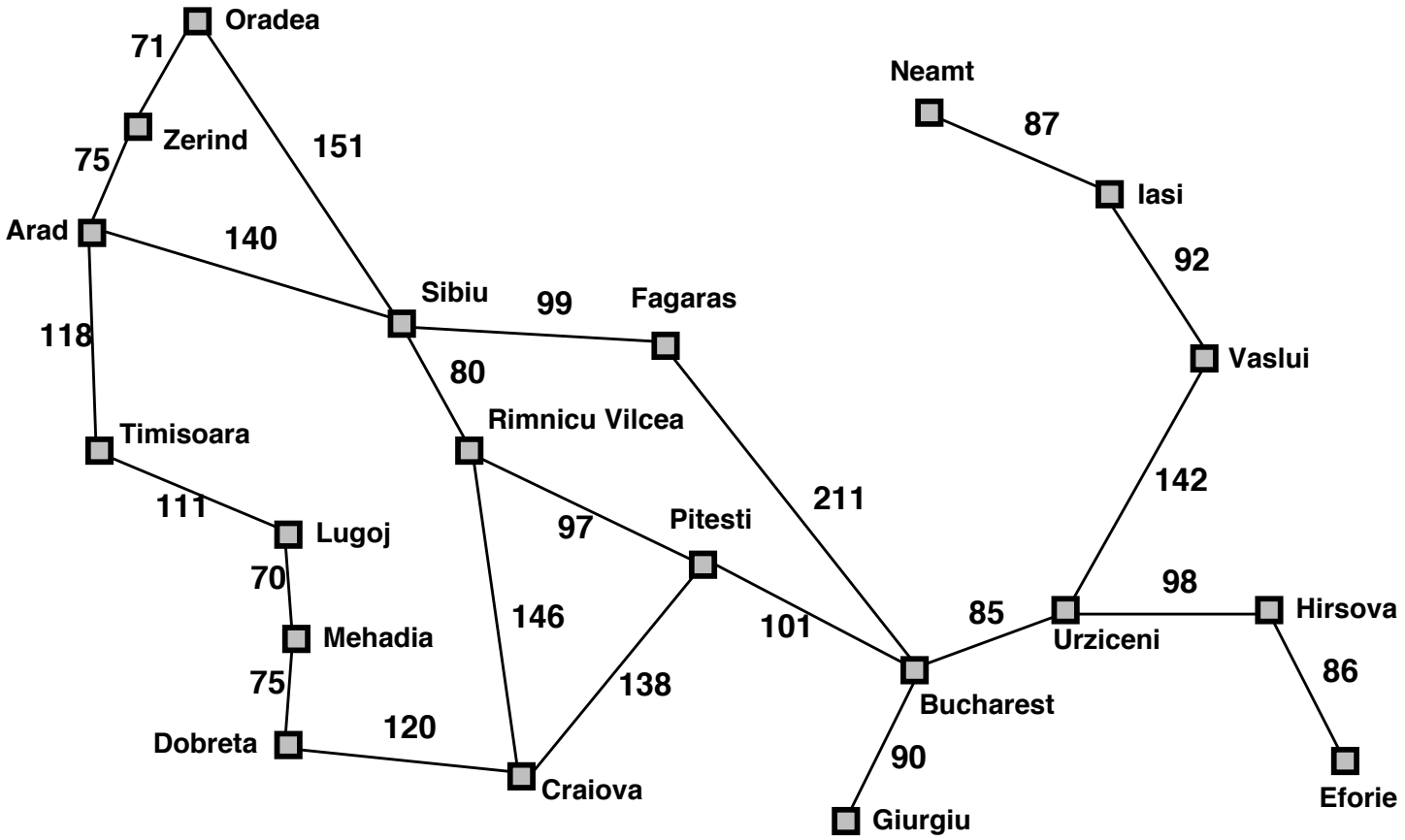
Queen Mary University of London



Exercise: IDS on holiday

Queen Mary University of London

- Execute IDS for the Romanian holiday problem, on paper



Analysing search algorithms for cost

Queen Mary University of London

- Iterative Deepening Search
  - Complete? Yes
  - Time? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
  - Space? $O(bd)$
  - Optimal? Only if step cost = 1
  - Pros: combines the benefits of BFS and DFS
    - optimal in path length and complete (like BFS)
    - modest memory requirements (like DFS)
    - can be modified to explore uniform-cost tree
  - Cons: some (perhaps many) states are expanded multiple times
    - Although this appears wasteful, the overhead is actually quite small for most cases

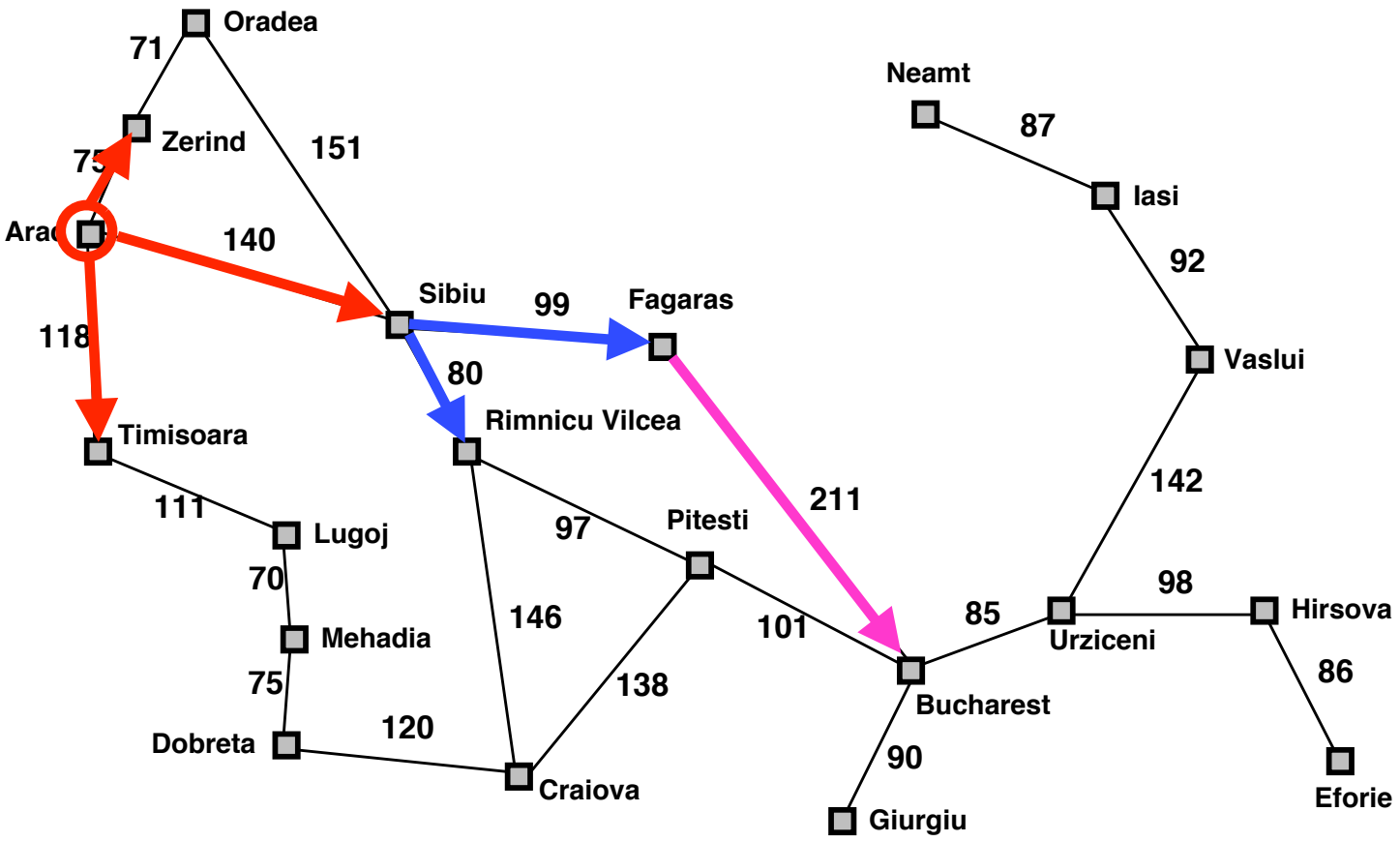
Informed Search

Queen Mary University of London

- *Informed search methods* use problem specific information about the state space to generate solutions more effectively
  - Sometimes called Best-First Search (but UCS is also sometimes called best-first)
- Extra knowledge is provided by a *heuristic* which returns a number describing the desirability of expanding a given node
  - an approximation since we usually don't know the best node to expand
  - nodes are expanded in order of desirability (like UCS)
  - the most desirable are processed first
- The aim is to find *low cost solutions*
  - typically evaluate the estimated cost of a solution and try to minimise it
  - incorporate an estimate of the cost from the state to the closest goal state
    - try to expand the node closest to the goal, or
    - try to expand the node on the least cost solution path

Greedy, or hill-climbing, search

Queen Mary University of London

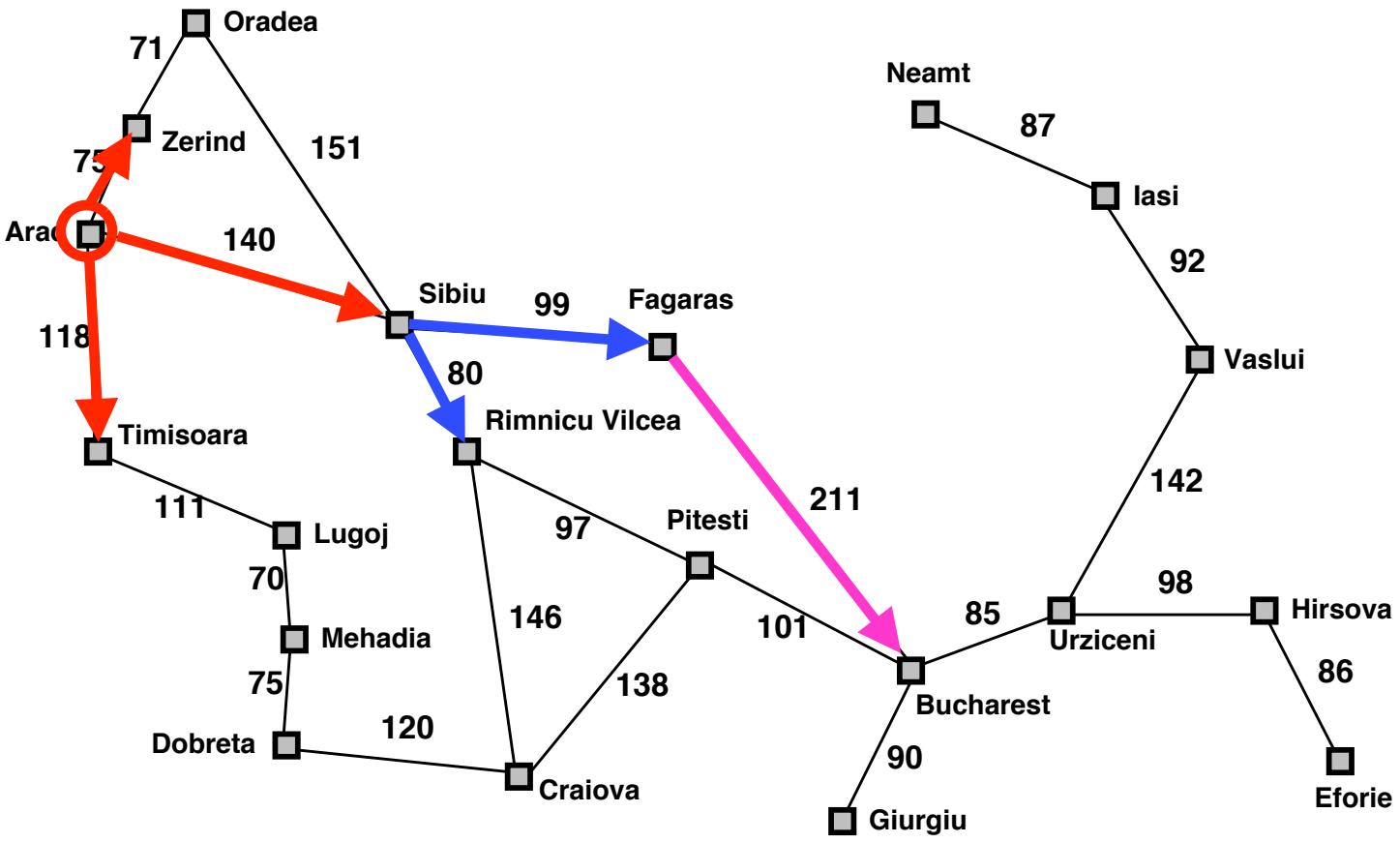


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy, or hill-climbing, search

Queen Mary University of London

- The simplest kind of informed search *estimates the cost* from the current node *to the solution* and uses that to sort its agenda
  - note the difference with UCS, where we use the *actual cost so far*



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, or hill-climbing, search

Queen Mary University of London

- Most greedy search has an agenda of length 1
  - all but the best node is thrown away, at every cycle
  - but it can be implemented with an agenda of any length
  - the shorter the agenda, the more likely the search is to get caught in local minima, where there is a temporary dip in the solution cost
    - throwing away solutions means you can't recover if you go astray
- In the route-finding problem, a good heuristic is the straight-line distance to the goal
  - "as the crow flies"
  - this idea, that a *geometrical distance* estimates a distance along a graph, is the basis of many heuristics
- But because it doesn't account for cost so far, it can lead us astray

Algorithm A

Queen Mary University of London

- Algorithm A combines the best of UCS and Greedy search
  - UCS had "cost so far"
  - Greedy had "estimated cost remaining" or *heuristic*
  - Algorithm A has "cost so far" + "estimated cost remaining"
- This is usually notated as $f(n) = g(n) + h(n)$
  - f is the overall cost/evaluation/utility function
  - g is the cost-so-far function
  - h is the heuristic (estimated cost remaining)
- Algorithm A
  - is complete
  - but can be sub-optimal in node expansion

Algorithm A*

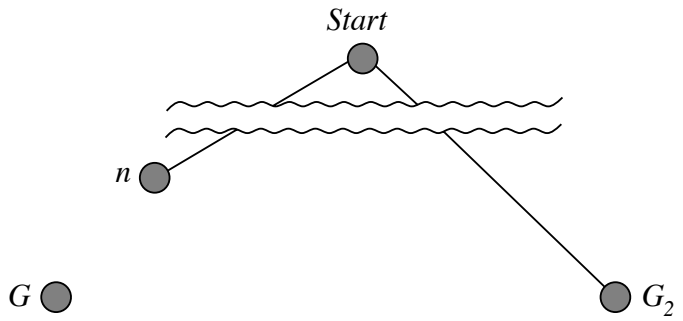
Queen Mary University of London

- Algorithm A* adapts Algorithm A by using an *admissible* heuristic
  - this means that it *never over-estimates* the distance to the closest solution
  - in any graph representation of space, geometric distance has this property
- A* is optimal
  - in the sense that we are guaranteed to find a solution if there is one (complete)
  - in the sense that the solution will have minimum cost
  - in the sense that the fewest possible nodes will be expanded, given this problem formulation (efficient)
    - NB re-formulating (a better heuristic) is likely to produce a better search
- Exercise
  - Using the data on the earlier slide, apply A* to the Arad-Bucharest route problem
    - in what order are the nodes expanded?

Analysing search algorithms for cost

Queen Mary University of London

- Proving optimality of A*
  - Suppose there is a suboptimal goal G_2 in the agenda
    - Let n be an unexpanded node on a shortest path to an optimal goal G
  - $f(G_2) = g(G_2) + h(G_2)$
    - but $h(G_2) = 0$, as it's a goal
  - $g(G_2) > g(G)$
    - because G_2 is sub-optimal
  - $g(G) \geq f(n)$
    - because h is admissible
  - Because $f(G_2) > f(n)$, A* will never select G_2 for expansion



- The more accurate h is, the fewer unnecessary nodes will be expanded

Analysing search algorithms for cost

Queen Mary University of London

- One way to characterise the quality of a heuristic is the *effective branching factor* (b^*)
  - if
    - the total number of nodes expanded by A* for a particular problem is N
    - the solution depth is d
  - then
    - b^* is the branching factor that a uniform tree of depth d would have to have in order to contain N nodes
  - $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$
    - e.g., if A* finds a goal at depth 5 using 52 nodes, the effective branching factor is 1.91
- Usually the effective branching factor exhibited by a given heuristic is fairly constant over a large range of problems
  - This can be used for empirical evaluation of a heuristic's usefulness

Exercise: The 8-Puzzle

Queen Mary University of London

- Choosing a heuristic function for a more abstract problem
  - the 8-puzzle
    - typical solution ≈ 20 steps
    - branching factor ≈ 3 (2 if space in corner, 4 in middle, 3 otherwise)
    - exhaustive search to depth 20 examines $3^{20} \approx 3.5 \times 10^9$ nodes
  - A* Heuristic must never over-estimate number of steps to goal
    - h_1 = number of tiles in wrong position
    - h_2 = sum of Manhattan/city-block distances of tiles from their goals

Choosing a heuristic for 8-puzzle

Queen Mary University of London

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Table comparing the search costs and effective branching factors for IDS and A* using h_1 and h_2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

- h_2 is better than h_1 which is better than uninformed search
- Always use a heuristic with higher values, so long as it doesn't over-estimate

Further Reading

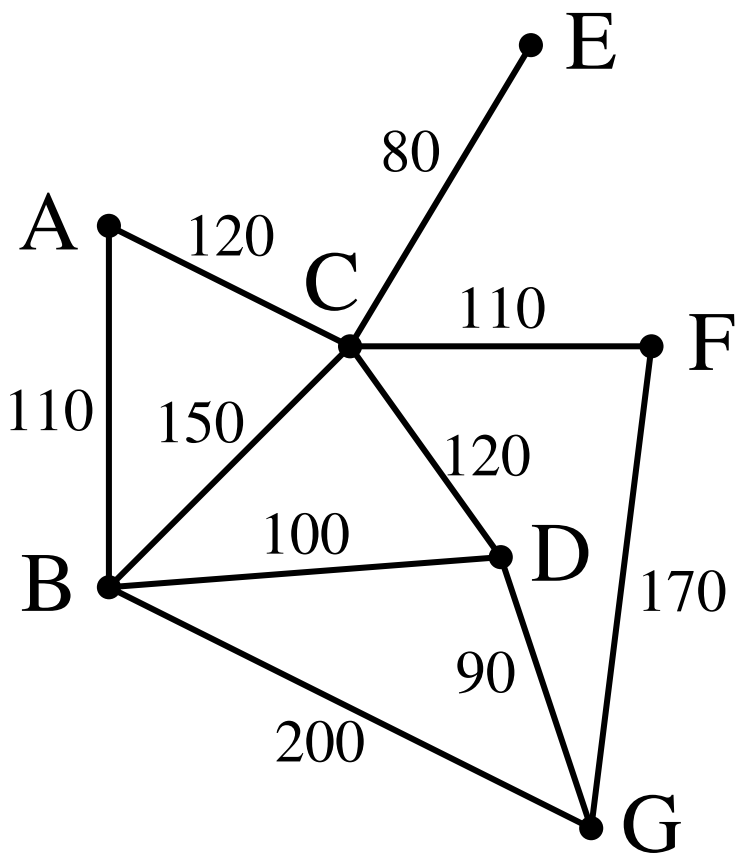
Queen Mary University of London

- In Chapters 3 & 4 of the textbook you can read more about search including
  - Bidirectional search
  - Searching with partial information
  - Hill-climbing search and simulated annealing
  - Genetic algorithms
  - Online search agents

Past **Exam**ple Question

Queen Mary University of London

Consider the problem of finding the cheapest flight routes between pairs of cities, where a map as shown to the right is given. Vertices represent cities, and are labelled by upper case letters. Edges represent direct flights between pairs of cities, and are labelled with the price of the flight (in pounds). Where there is a direct flight between a pair of cities, it operates in both directions and it has the same price in each direction. The required utility function considers only the cost, ignoring the number of separate flights, timetabling of flights, reputation of airline, etc.



Past **Exam**ple Question

Queen Mary University of London

a) Using depth-first search, what are the first 3 nodes expanded to find a path from F to B? [2 marks]

b) Why is depth-first search not suitable for this problem? How can depth-first search be modified to make it suitable for this problem? [4 marks]

c) What are the first three nodes expanded to find a path from F to B using breadth-first search? [2 marks]

d) What advantages does breadth-first search have over depth-first search in general (i.e. not only for route-finding problems)? (Explain any terms you use.) [4 marks]

e) What disadvantages does breadth-first search have for this route-finding problem? [4 marks]

f) Execute uniform-cost search to find a path from G to E. Show your working including the order of node expansion and the agenda at each step. What is the path and its cost found by the algorithm? [6 marks]

g) Would A* be a better algorithm for this problem? Why/why not? [3mks]