Artificial Intelligence

INFORMED SEARCH STRATEGIES

Nguyễn Ngọc Thảo – Nguyễn Hải Minh

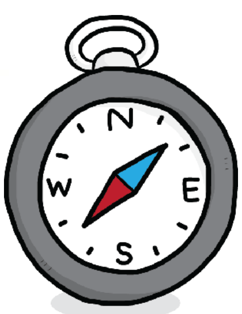
{nnthao, nhminh}@fit.hcmus.edu.vn

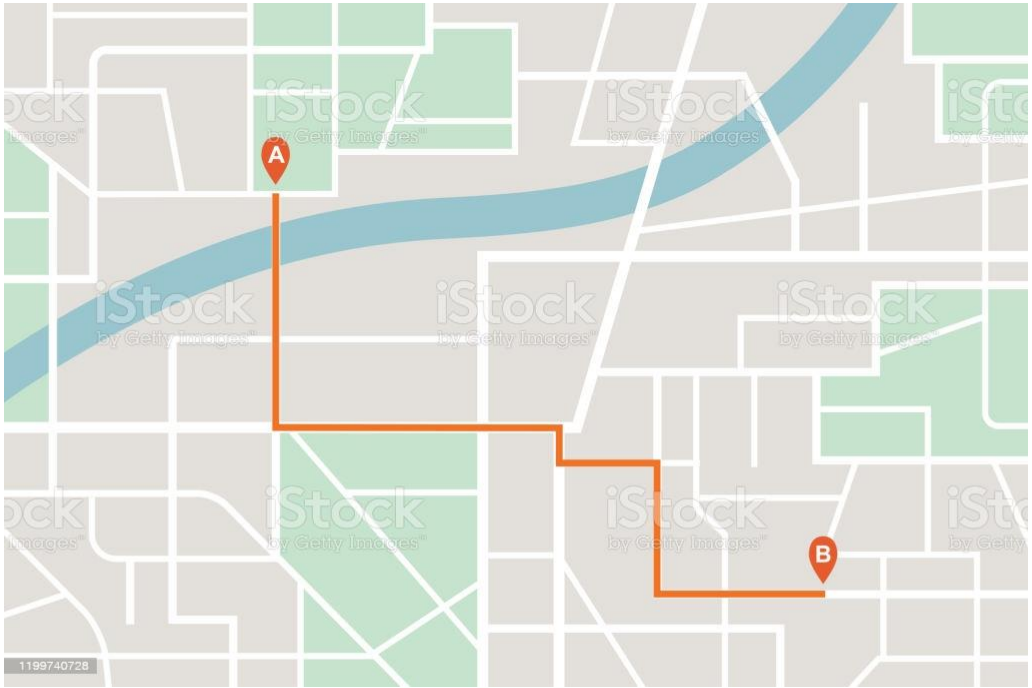
Outline

- Informed (Heuristic) search strategies
- Best-first search
- Greedy best-first search
- A* search
- Memory-bounded heuristic search
- Heuristic functions

Informed (Heuristic) search strategies

- Use problem-specific knowledge beyond the definition of the problem itself
- Find solutions more efficiently
- Provide significant speed-up in practice



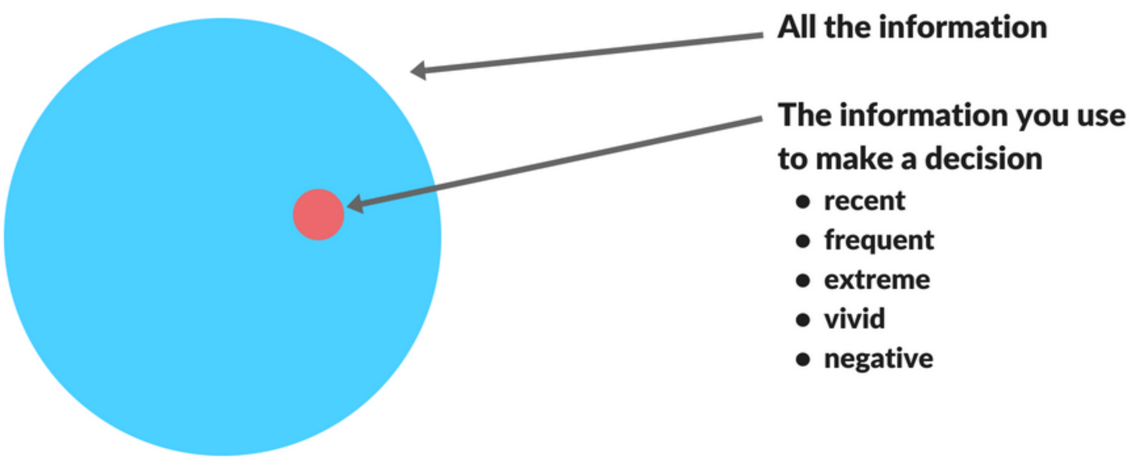


What are heuristics?

- Additional knowledge of the problem is imparted to the search algorithm using **heuristics**.
- A heuristic is any practical approach to problem solving sufficient for reaching an immediate goal where an optimal solution is usually impossible.
  - Not guaranteed to be optimal, perfect, logical, or rational
  - Speed up the process of finding a satisfactory solution
  - Ease the cognitive load of making a decision

Heuristics: An example

- Availability heuristic: what comes to mind quickly seems to be significant



Heuristics: An example

- Representativeness heuristic: estimate the likelihood of an event by comparing it to a prototype already exists in mind.

The portrait of an old woman
who is warm and caring
with a great love of children

Informed search strategies

- Best-first search
- Greedy best-first search
- A*
- RBFS
- SMA*

Best-first search

- A best-first search algorithm can be either a TREE-SEARCH or GRAPH-SEARCH instance.
- A node is selected for expansion based on an **evaluation function**, $\boldsymbol{f(n)}$.
    - Node with the lowest $f(n)$ is expanded first
- The choice of f determines the search strategy.

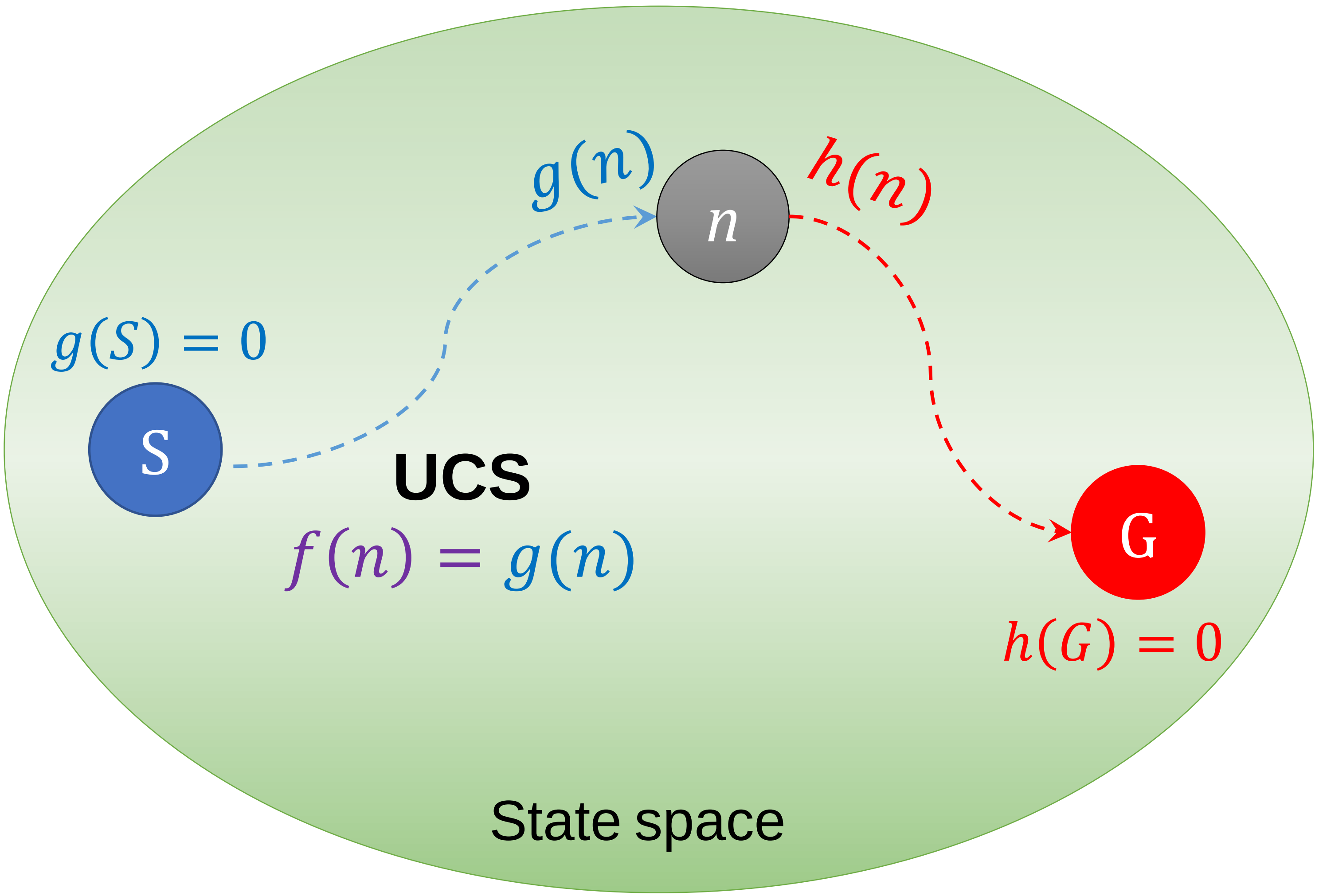
Heuristic function

- Most best-first algorithms include a **heuristic function $\boldsymbol{h(n)}$** as a component of f.

$\boldsymbol{h(n)}$	estimated cost of the cheapest path from the state at node n to a goal

- Unlike $g(n)$, $h(n)$ depends only on the state at that node
- Assumption of $h(n)$
    - Arbitrary, nonnegative, problem-specific functions
    - Constraint: if n is a goal node, then $\boldsymbol{h(n) = 0}$

Cost function vs. Heuristic function

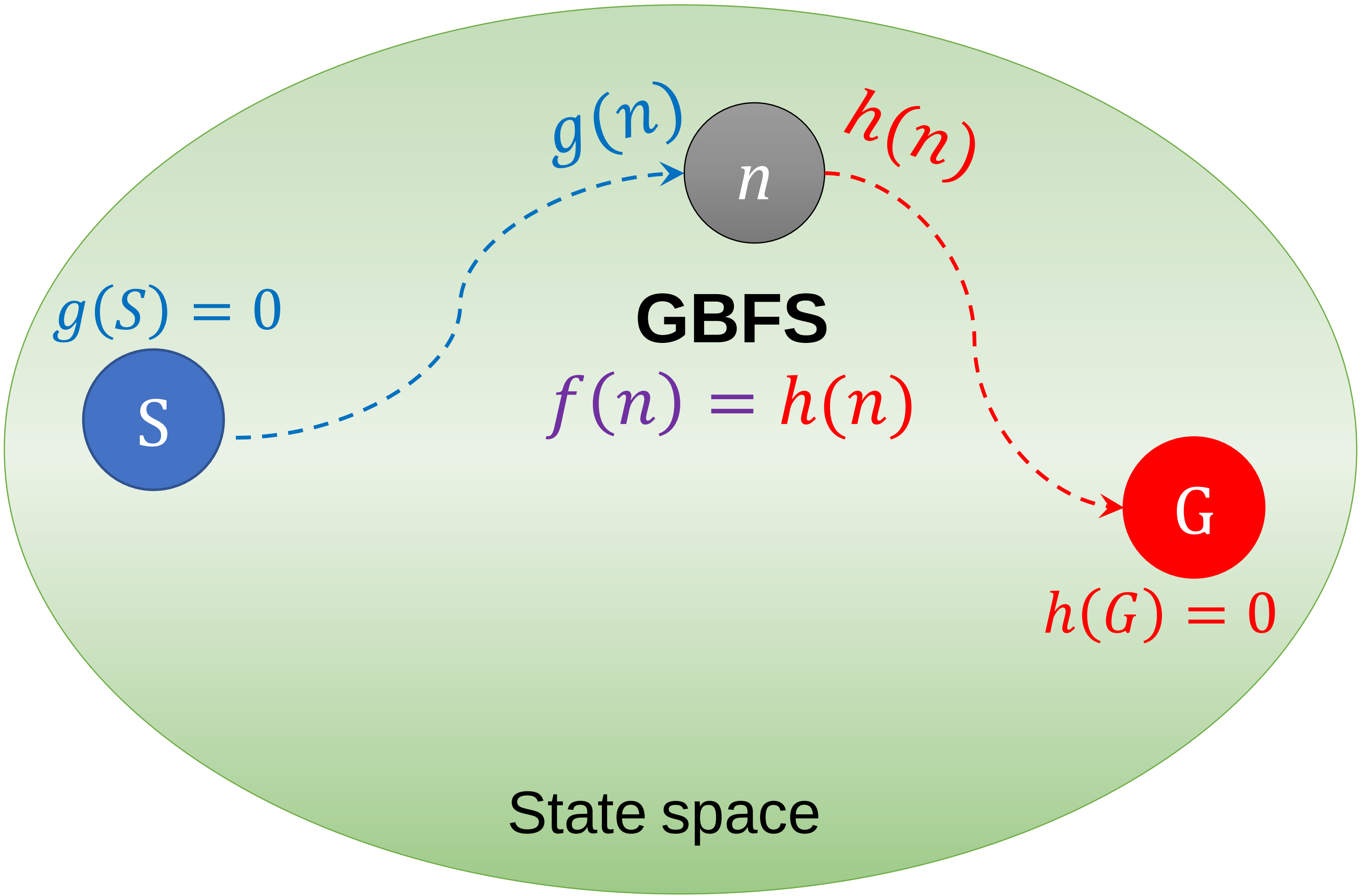


Greedy best-first search

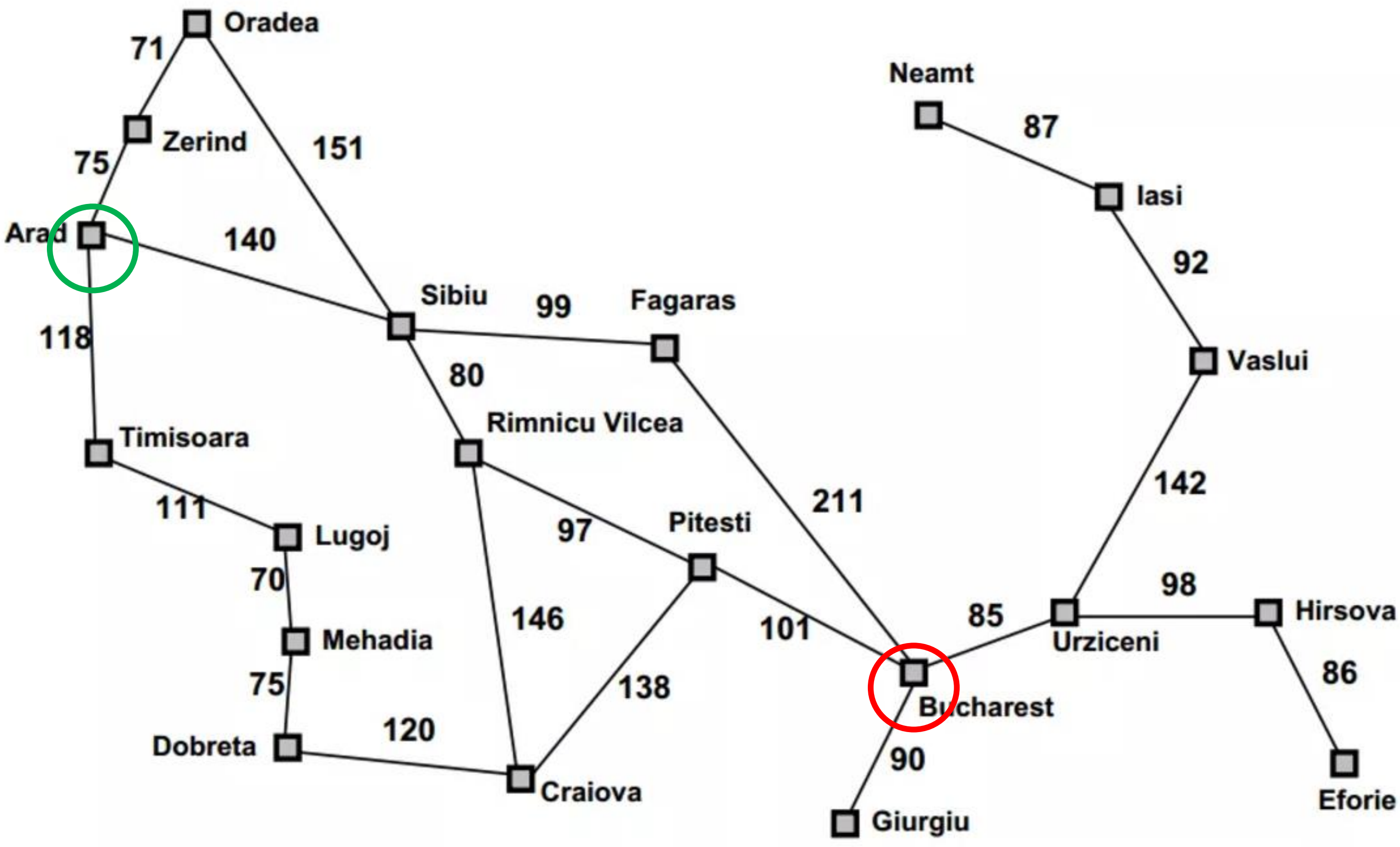
Greedy best-first search

- Expand the node that appears to be closest to goal using

$$\boldsymbol{f(n) = h(n)}$$



Straight-line distance heuristic h_{SLD}

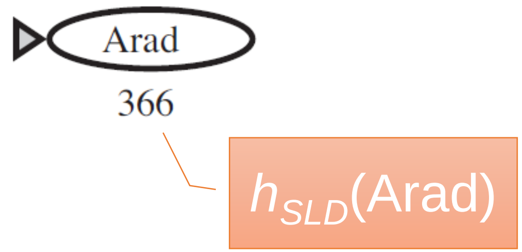


Straight–line distance to Bucharest

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: An example

(a) The initial state



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

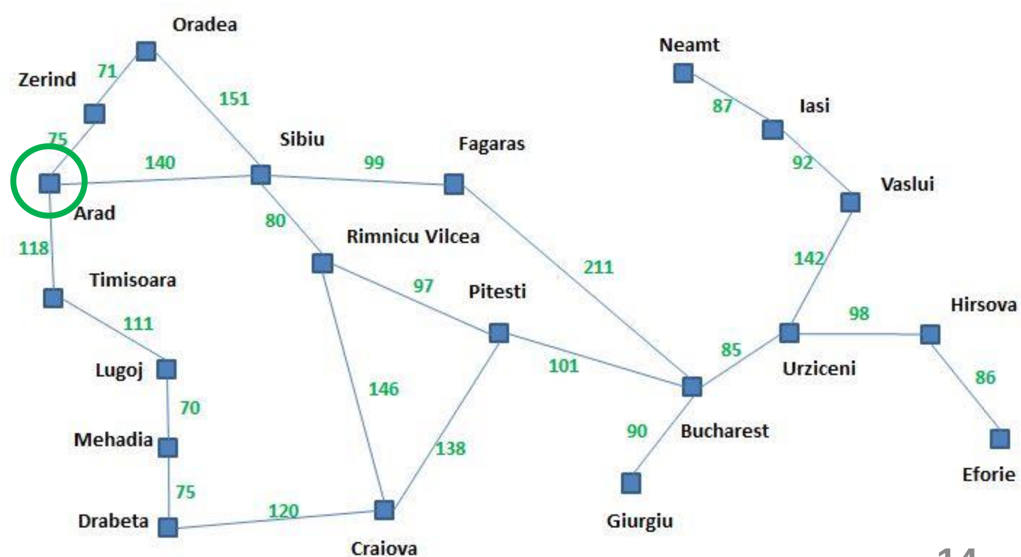
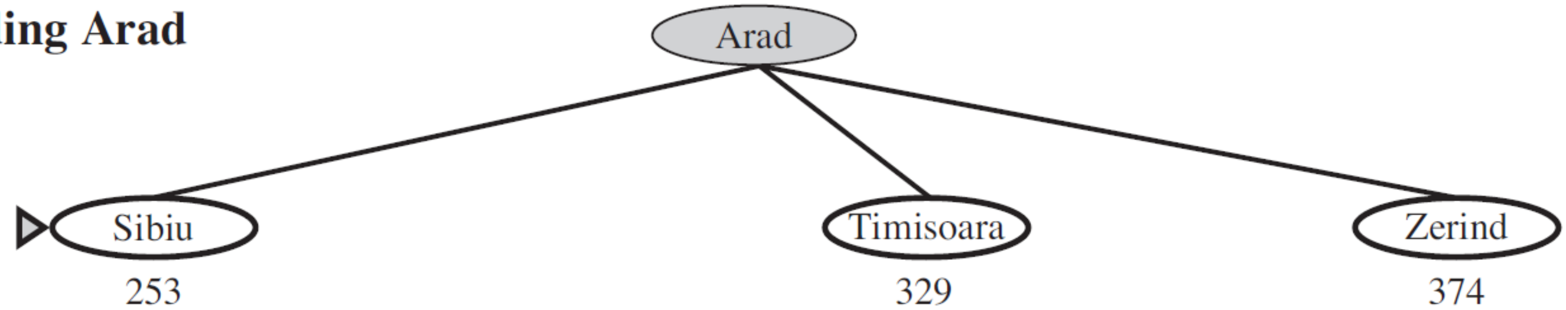


Figure A simplified road map of part of Romania.

Greedy best-first search: An example

(b) After expanding Arad



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

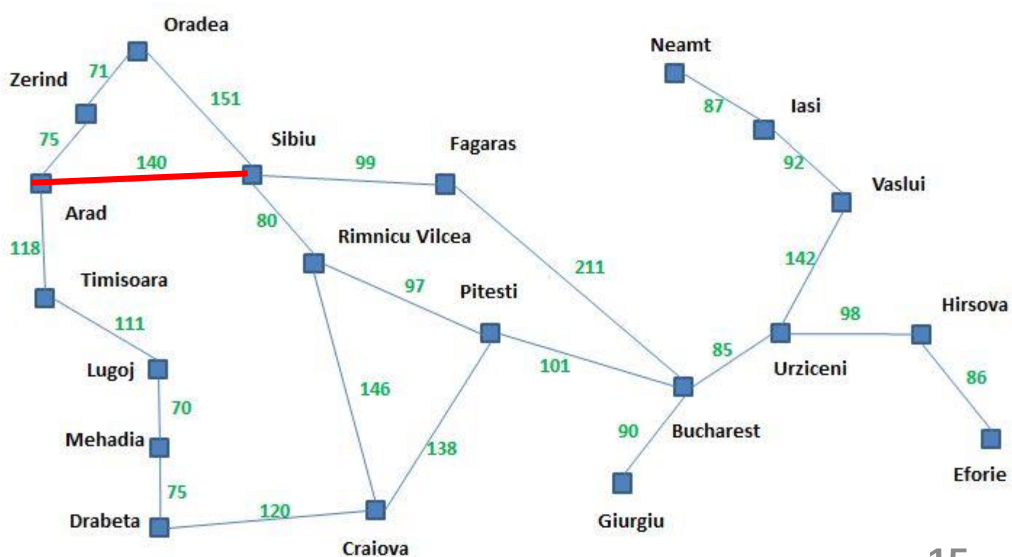
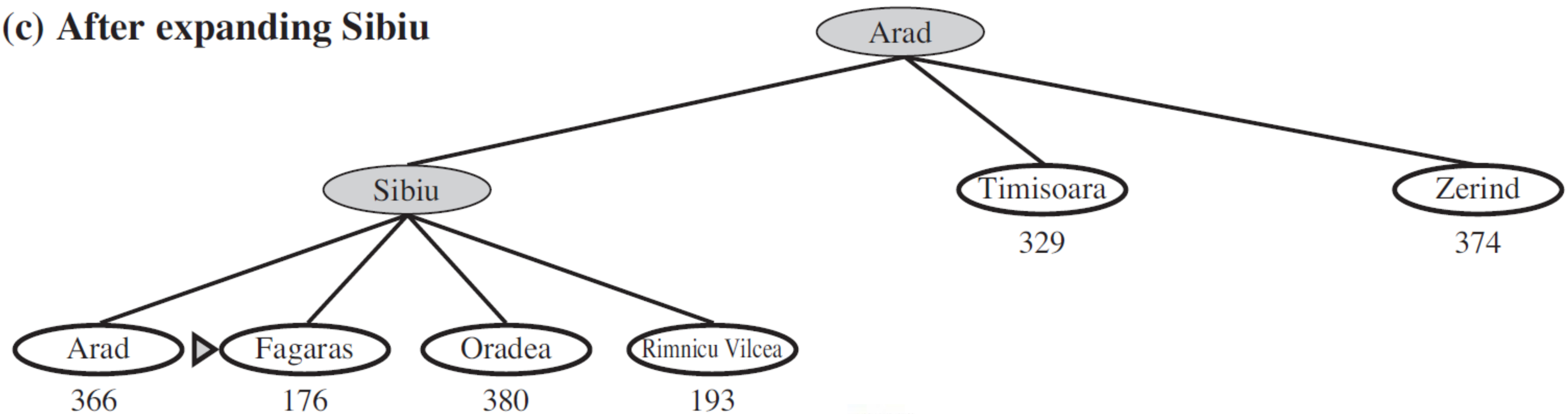


Figure A simplified road map of part of Romania.

Greedy best-first search: An example

(c) After expanding Sibiu



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

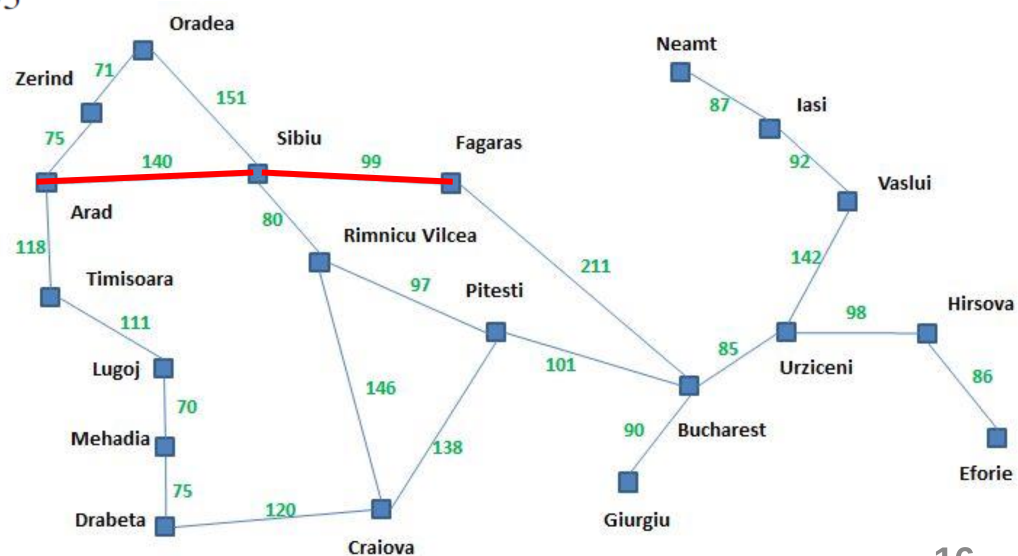
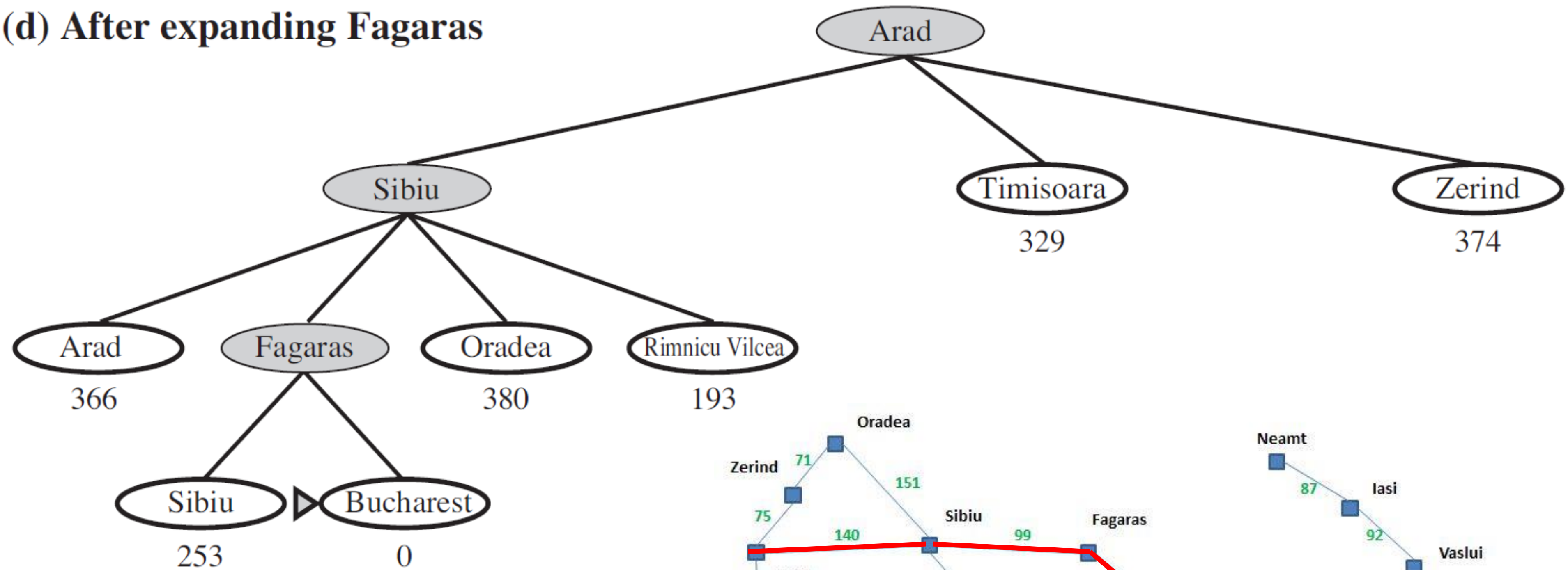


Figure A simplified road map of part of Romania.

Greedy best-first search: An example

(d) After expanding Fagaras



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

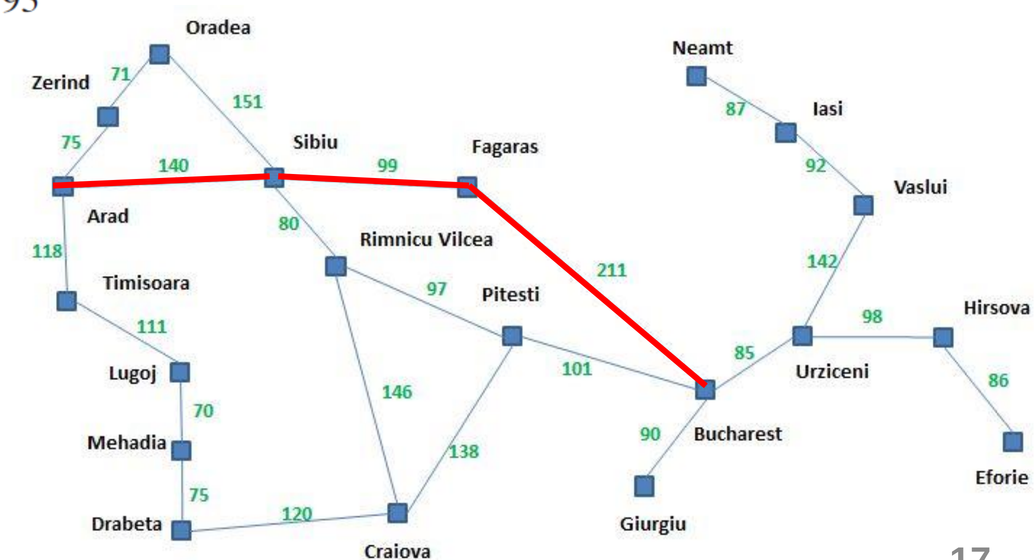


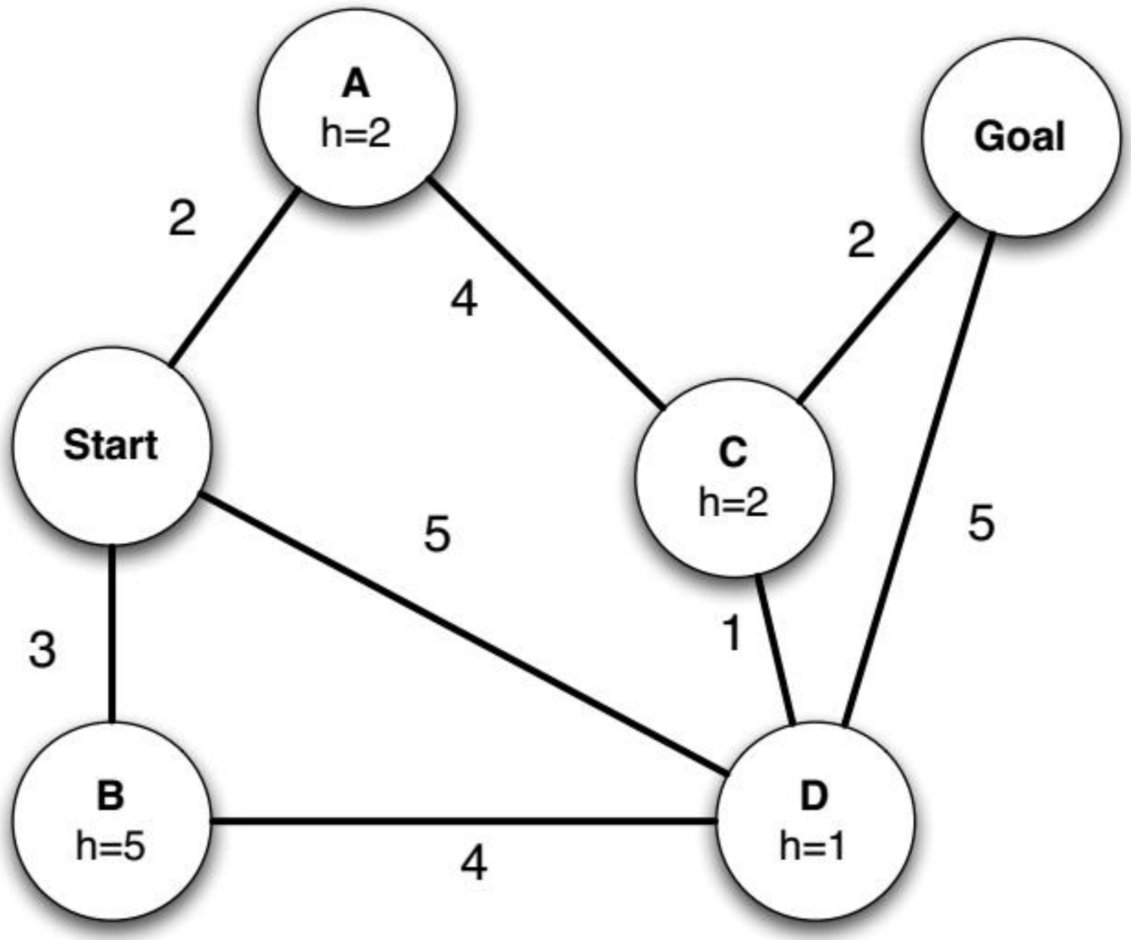
Figure A simplified road map of part of Romania.

An evaluation of GBFS (graph-search)

- Completeness
  - YES – if it is a graph-search instance in finite state spaces
- Time complexity
  - $O(b^m) \to$ reduced substantially with a good heuristic
- Space complexity
  - $O(b^m)$ – keeps all nodes in memory
- Optimality
  - NO

Quiz 01: Greedy best-first search

- Work out the order in which states are expanded, as well as the path returned by graph search. Assume ties resolve in such a way that states with earlier alphabetical order are expanded first.



A* Search

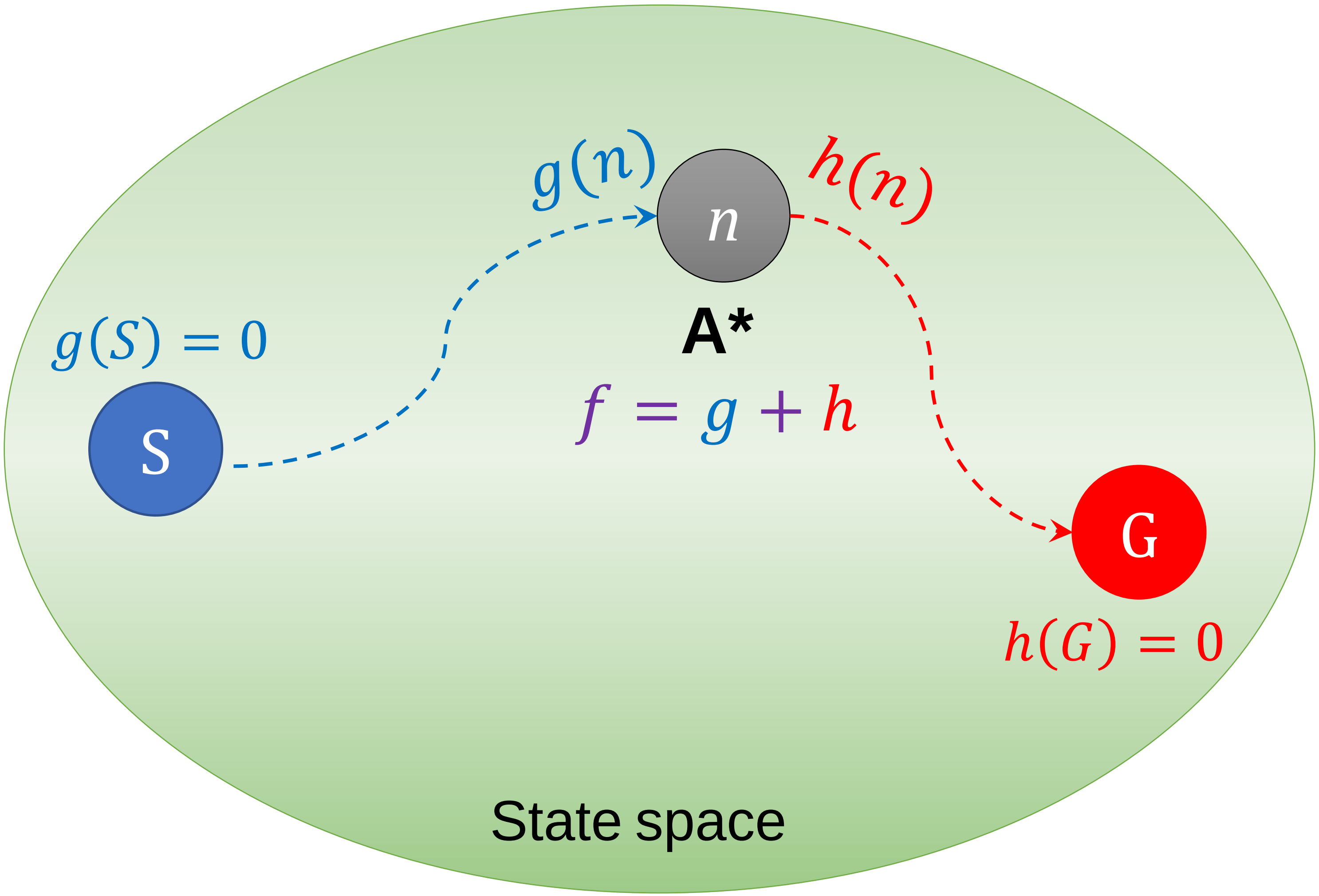
A* search

- The most widely known form of best-first search
- Use heuristic to guide search, but not only
- Avoid expanding paths that are already expensive
- Ensure to compute a path with minimum cost

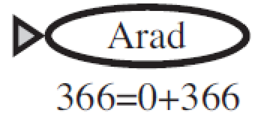
- Evaluate nodes by $\boldsymbol{f(n) = g(n) + h(n)}$
  - where $g(n)$ is the cost to reach the node n and $h(n)$ is the cost to get from n to the goal
  - $f(n) =$ estimated cost of the cheapest solution through n

.

A* search



(a) The initial state



$f = g + h$

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

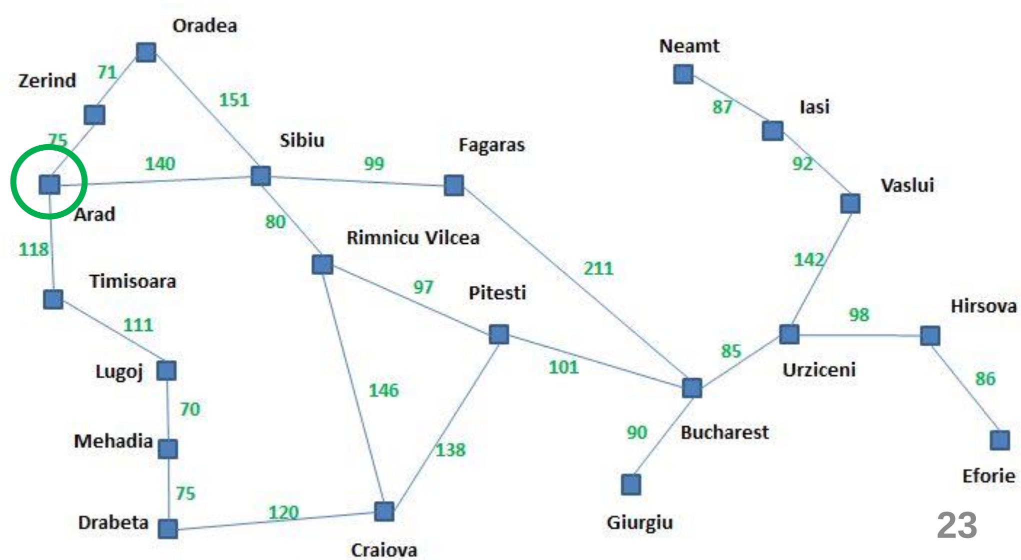
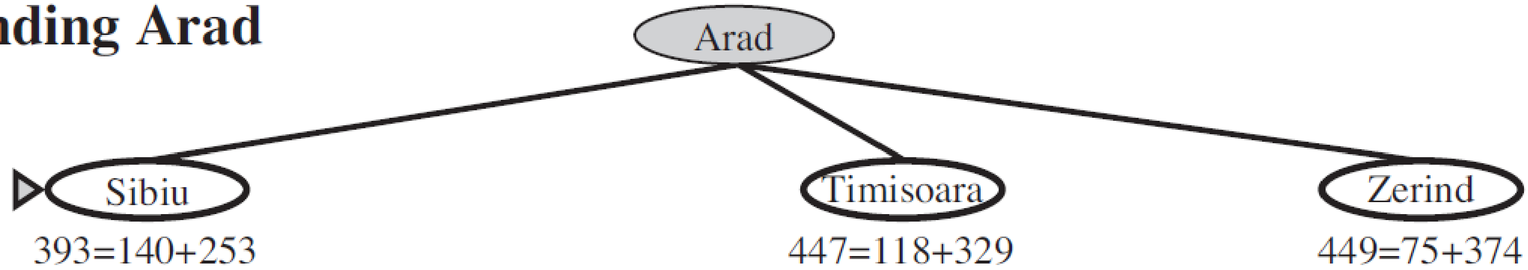


Figure A simplified road map of part of Romania.

(b) After expanding Arad



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

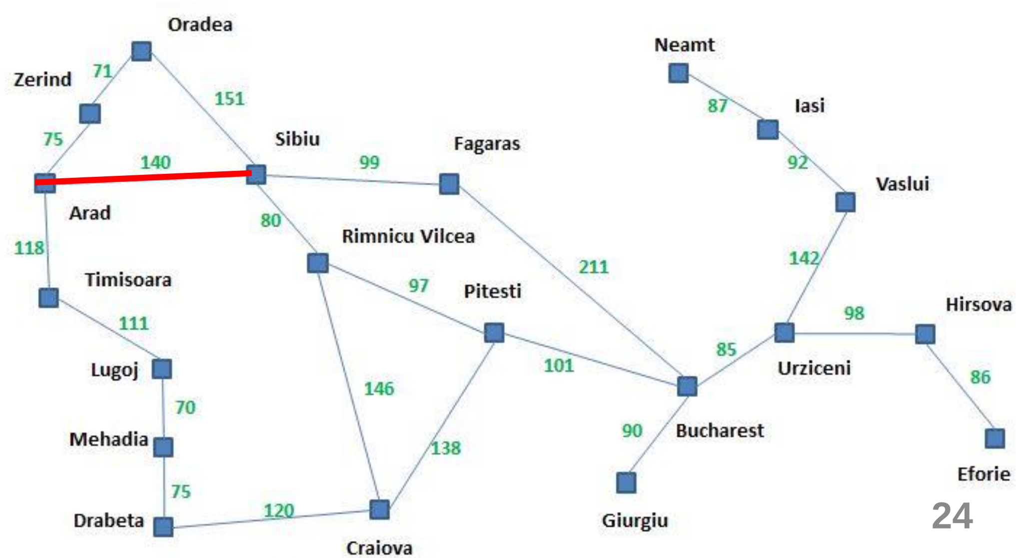
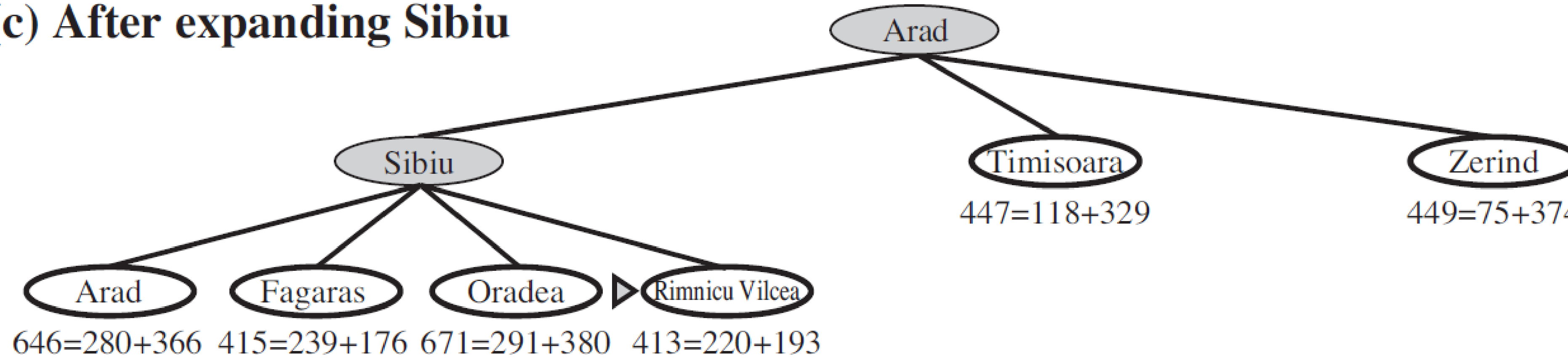


Figure A simplified road map of part of Romania.

(c) After expanding Sibiu



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

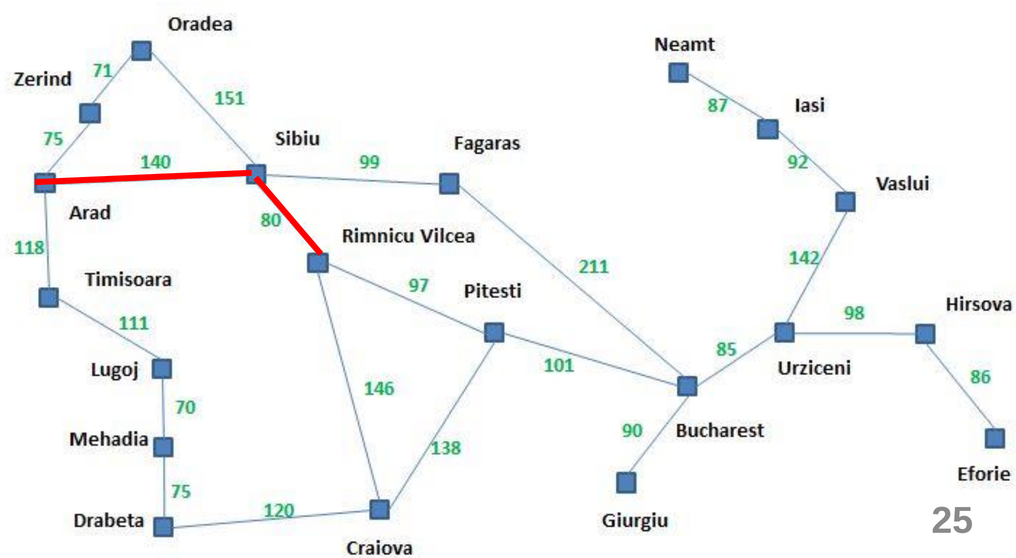
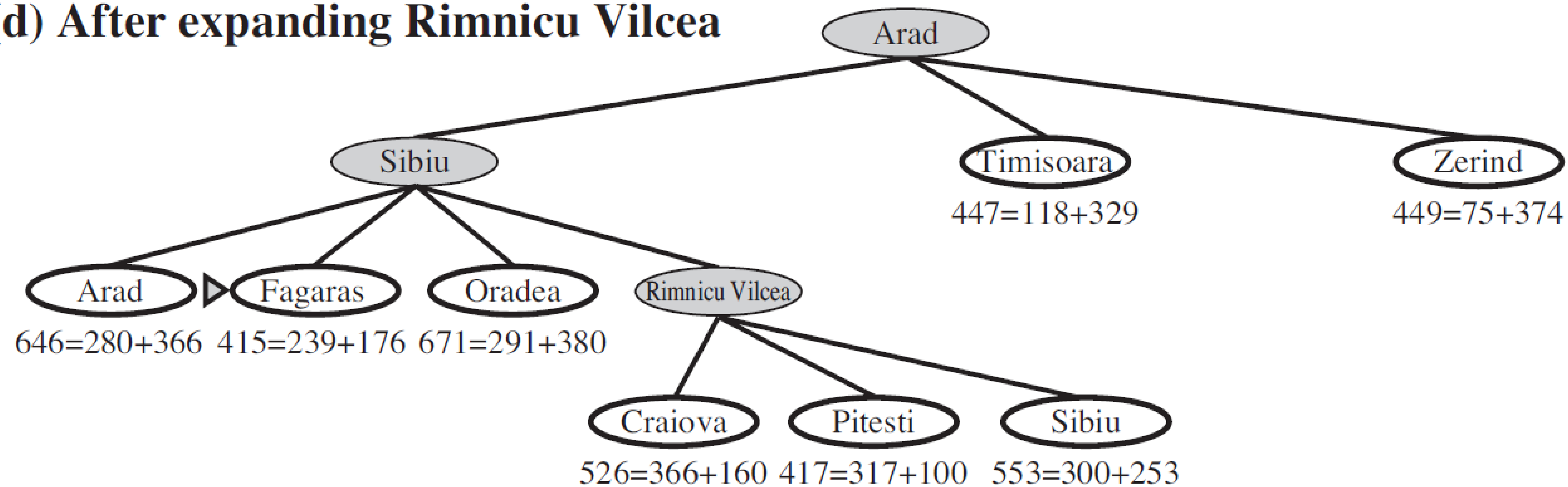


Figure A simplified road map of part of Romania.

(d) After expanding Rimnicu Vilcea



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

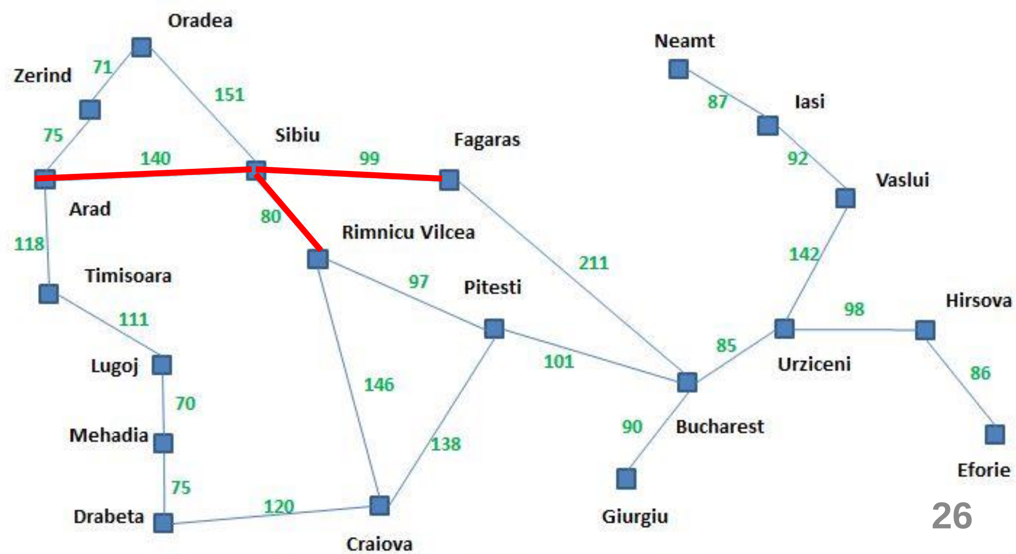
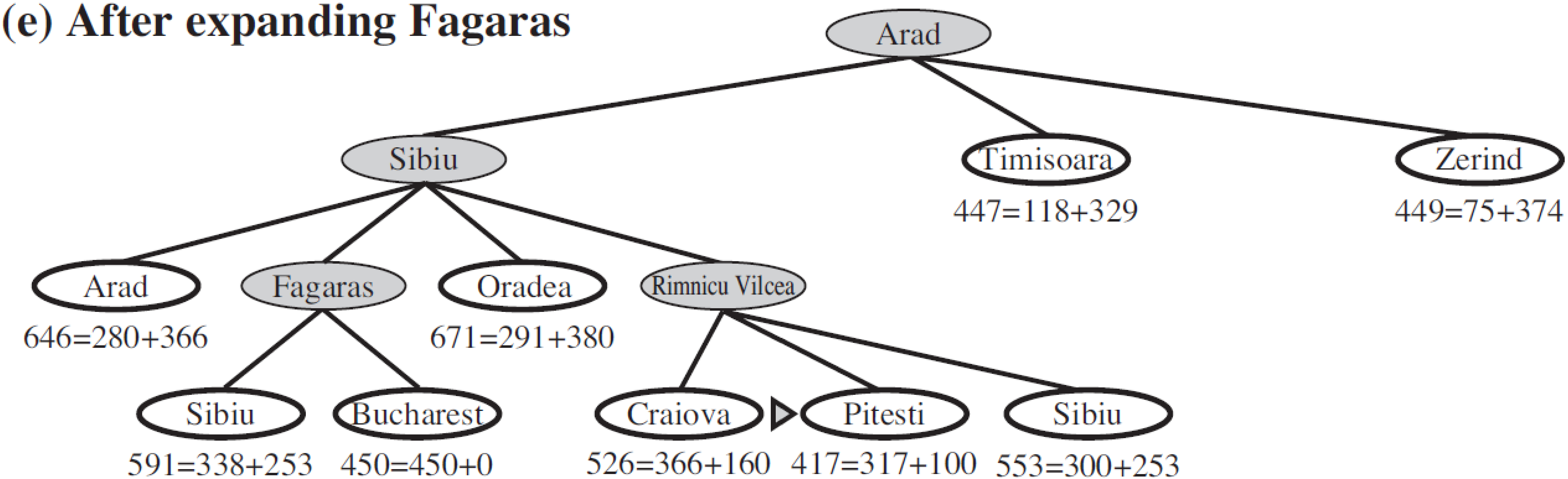


Figure A simplified road map of part of Romania.

(e) After expanding Fagaras



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

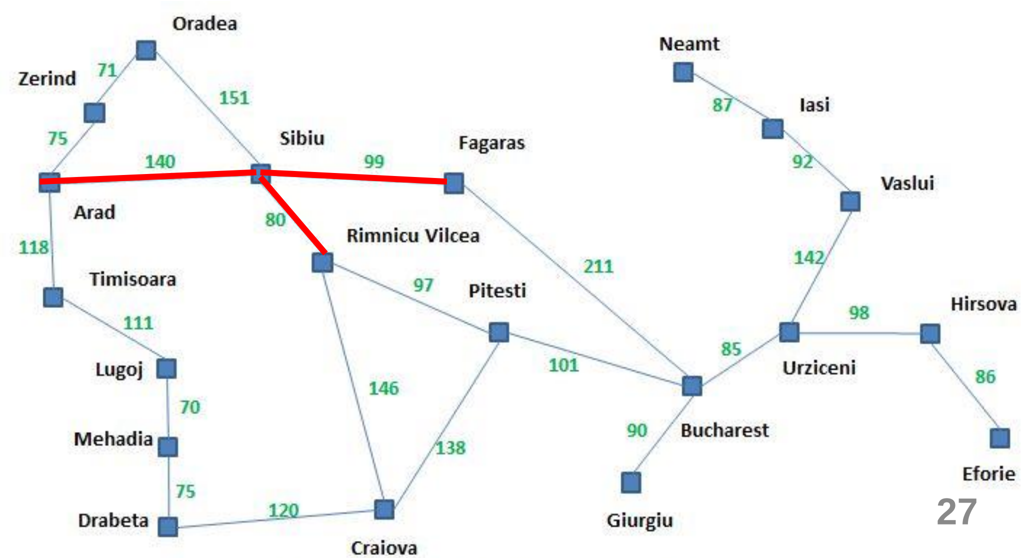
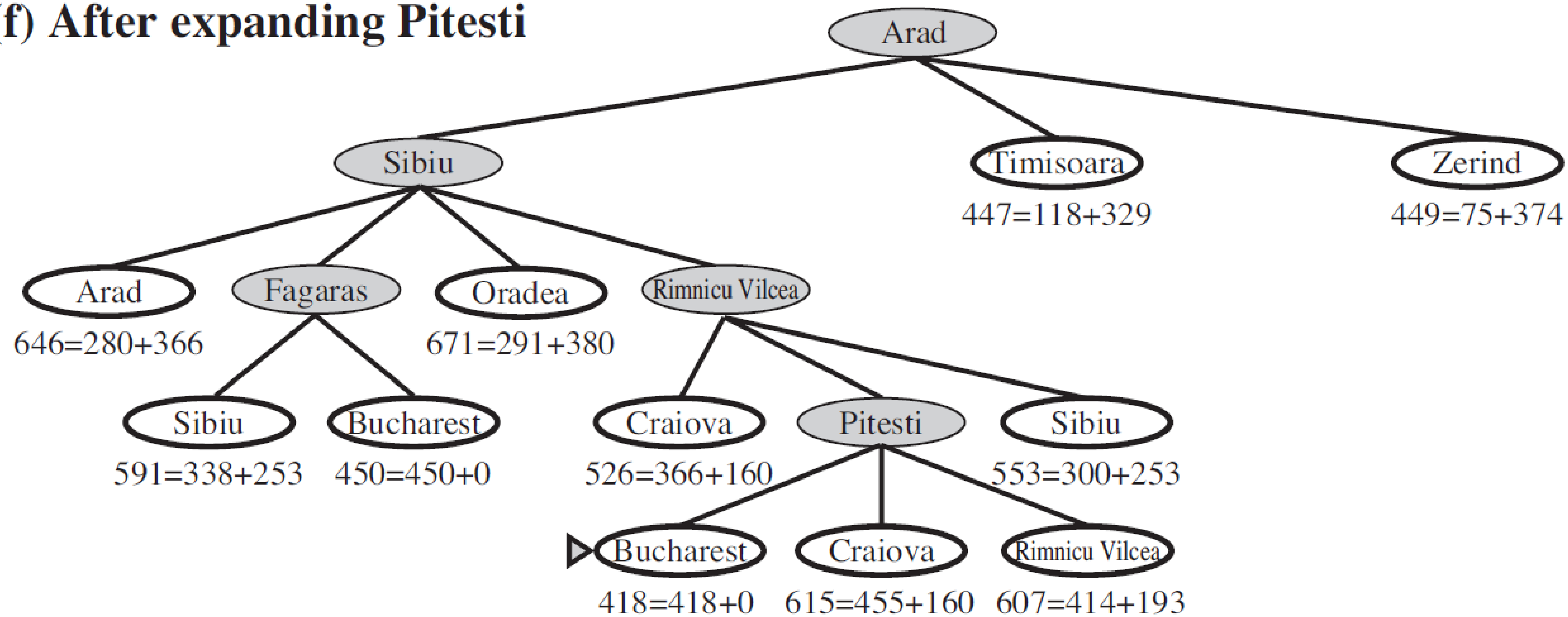


Figure A simplified road map of part of Romania.

(f) After expanding Pitesti



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

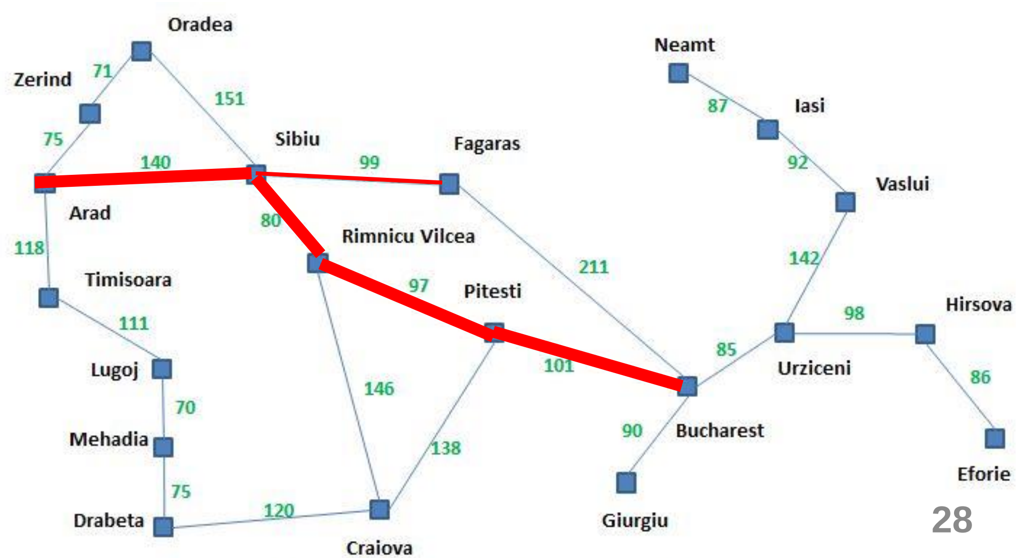
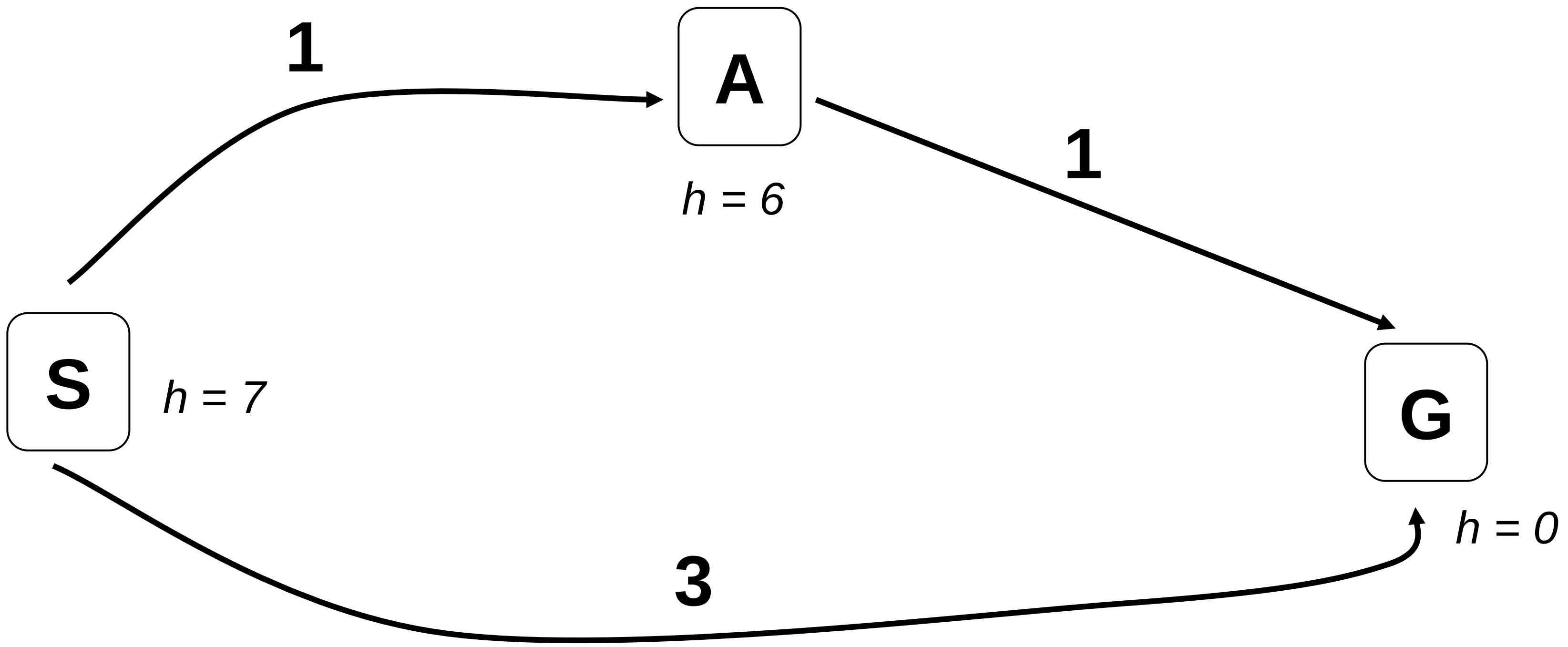


Figure A simplified road map of part of Romania.

An evaluation of A* (graph-search)

- Completeness
  - YES if all step costs exceed some finite ϵ and if b is finite
  - (review the condition for completeness of UCS)
- Optimality
  - YES – with conditions on heuristic being used
- Time complexity
  - Exponential
- Space complexity
  - Exponential (keep all nodes in memory)

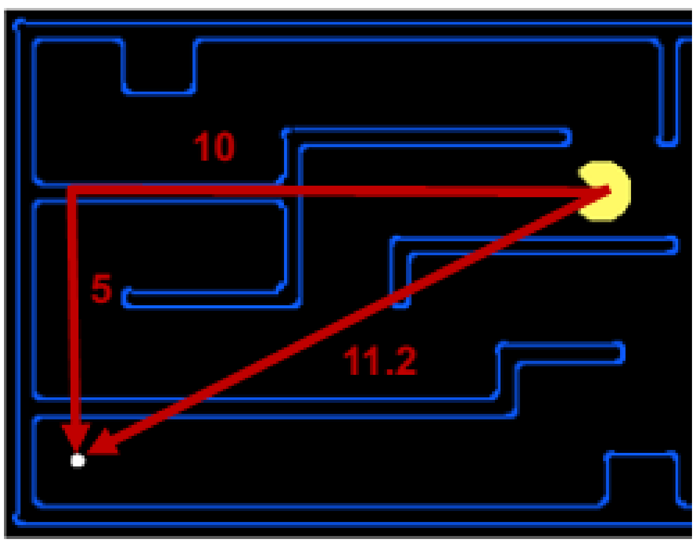
A* is not always optimal...



In what conditions, A* is optimal?

Conditions for optimality: Admissibility

- $h(n)$ must be an admissible heuristic
  - Never overestimate the cost to reach the goal $\rightarrow$ optimistic
  - E.g., the straight-line distance h_{SLD}



Admissible heuristics for 8-puzzle

- *h(n)* = number of misplaced numbered tiles

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

State n

$h(n) = 6$

1	2	3
4	5	6
7	8	

Goal state G

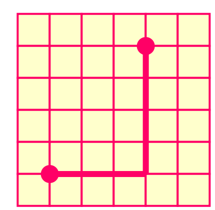
- *h(n)* = sum of the (Manhattan) distance of every numbered tile to its goal position

1		5
2	6	3
7	4	8

$h(n) = 9$

1	2	3
4	5	6
7	8	

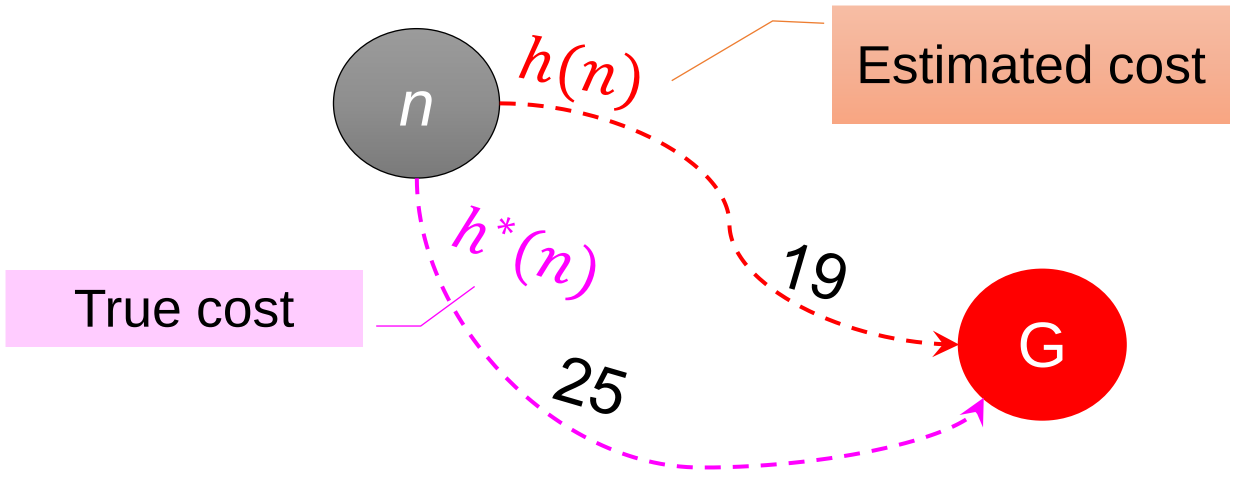
$h = 0 + 2 + 1 + 2 + 2 + 1 + 0 + 1$



$d = dx + dy$

Conditions for optimality: Admissibility

- $h(n)$ is admissible if for every node n, $\boldsymbol{h(n) \leq h^*(n)}$
  - where $h^*(n)$ is the true cost to reach the goal state from n

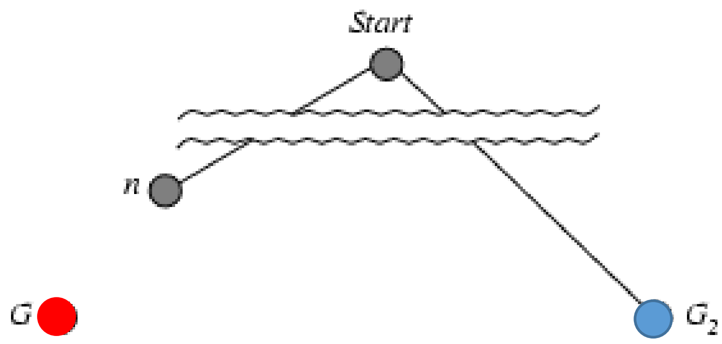


- Hence, $f(n)$ never overestimates the true cost of a solution along the current path through n.
  - $g(n)$ is the actual cost to reach n along the current path

Conditions for optimality: Admissibility

If $h(n)$ is admissible, A* using **TREE-SEARCH** is optimal

- Suppose some suboptimal goal G_2 has been generated and is in the frontier.
- Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$

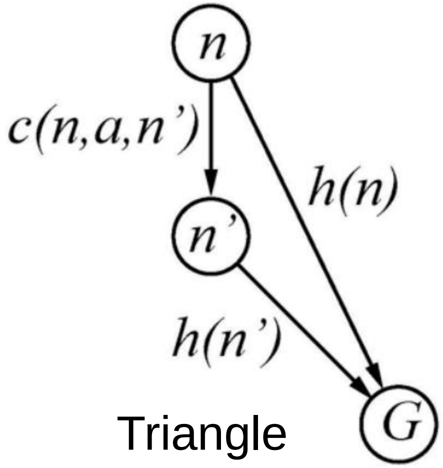
  $f(G_2) > f(G)$ (1)
- $h(n) \leq h^*(n)$ since *h* is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$

  $f(n) \leq f(G)$ (2)
- From (1), (2): $f(G_2) > f(n) \rightarrow$ A* will never select G_2 for expansion

Conditions for optimality: Consistency

- Admissibility is insufficient for graph search.
  - The optimal path to a repeated state could be discard if it is not the first one selected.
- $h(n)$ is consistent if for every node n, every successor n' of n generated by any action a,

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



Triangle inequality

- Every consistent heuristic is also admissible.

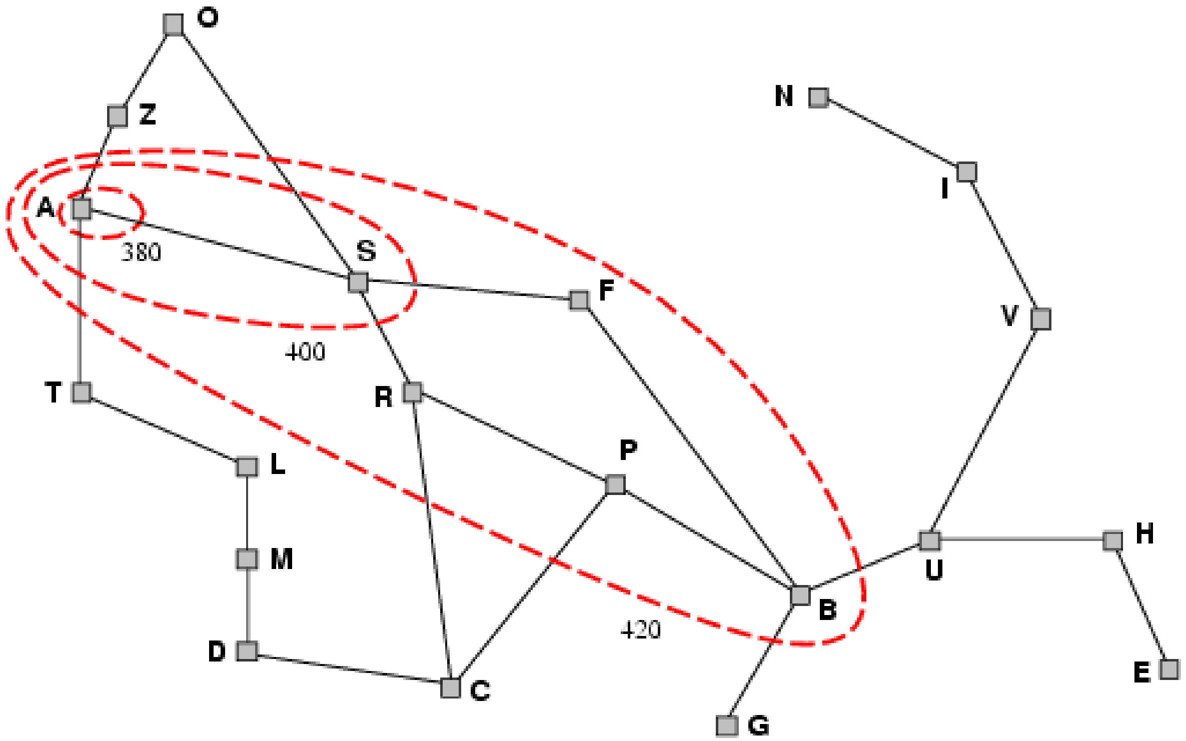
Conditions for optimality: Consistency

If $h(n)$ is consistent, A* using **GRAPH-SEARCH** is optimal

- If $h(n)$ is consistent, the values of $f(n)$ along any path are non-decreasing.
  - Suppose n' is a successor of $n \rightarrow g(n') = g(n) + c(n, a, n')$
  - $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$
- Whenever A* selects a node n for expansion, the optimal path to that node has been found.
  - Proof by contradiction: There would have to be another frontier node n' on the optimal path from the start node to n (by the graph separation property)
  - f is nondecreasing along any path $\rightarrow f(n') < f(n) \rightarrow n'$ would have been selected first

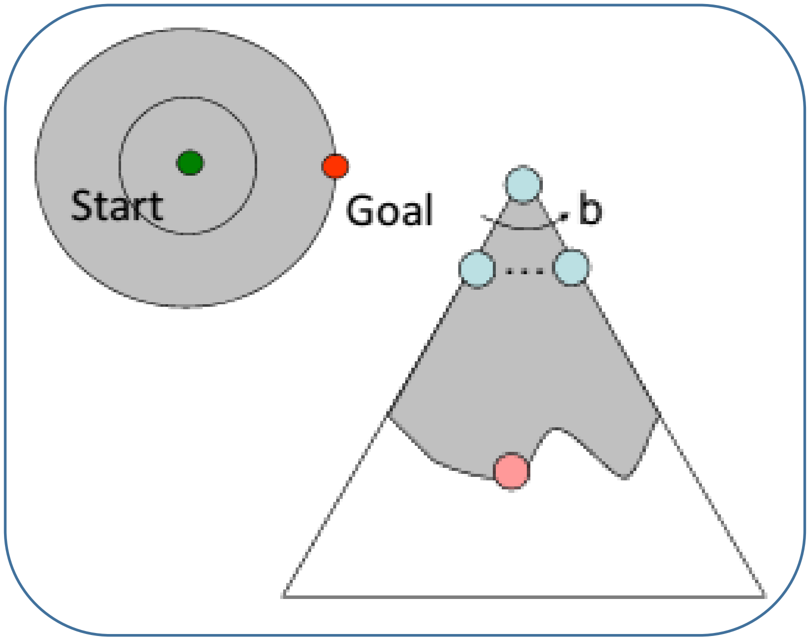
Contours of A* search

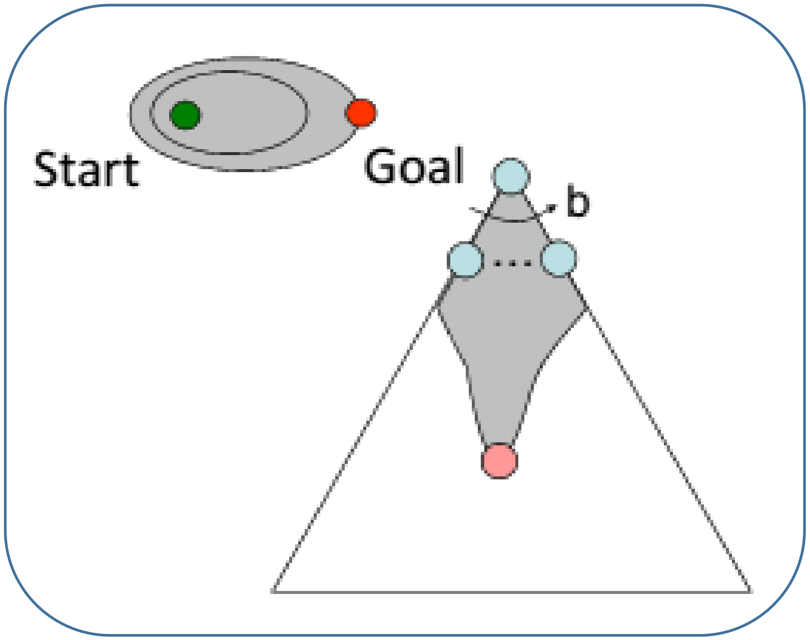
- A* expands nodes in order of increasing f-value
- Gradually adds "f-contours" of nodes such that contour i has all nodes with $f = f_i$ where $f_i < f_{i+1}$
- A* will expand all nodes with costs $f(n) < C^*$



A* contours vs. UCS contours

- The bands of UCS will be “circular” around the start state.





- The bands of A*, with more accurate heuristics, will stretch toward the goal state and become more narrowly focused around the optimal path.

Comments on A*: The good

- Never expand nodes with $f(n) > C^*$
  - All nodes like these are **pruned** while still guaranteeing optimality
- Optimally efficient for any given consistent heuristic
  - No other optimal algorithm is guaranteed to expand fewer nodes

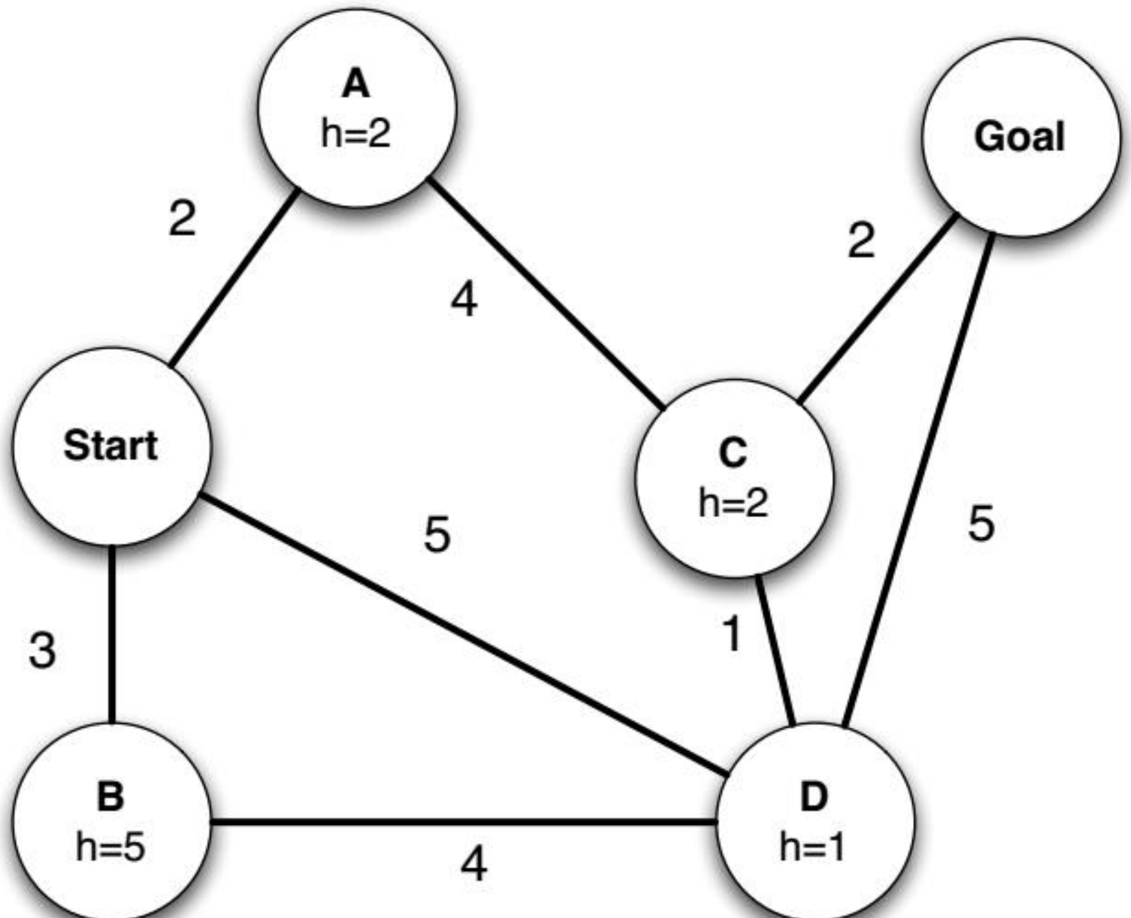
Comments on A*: The bad

- A* expands all nodes with $f(n) < C^*$ (and possibly some nodes with $f(n) = C^*$) with before selecting a goal node.
  - This can still be exponentially large
  - A* usually runs out of space before it runs out of time
- Exponential growth will occur unless error in $h(n)$ grows no faster than log(true path cost)
  - In practice, error is usually proportional to true path cost (not log)
  - So exponential growth is common

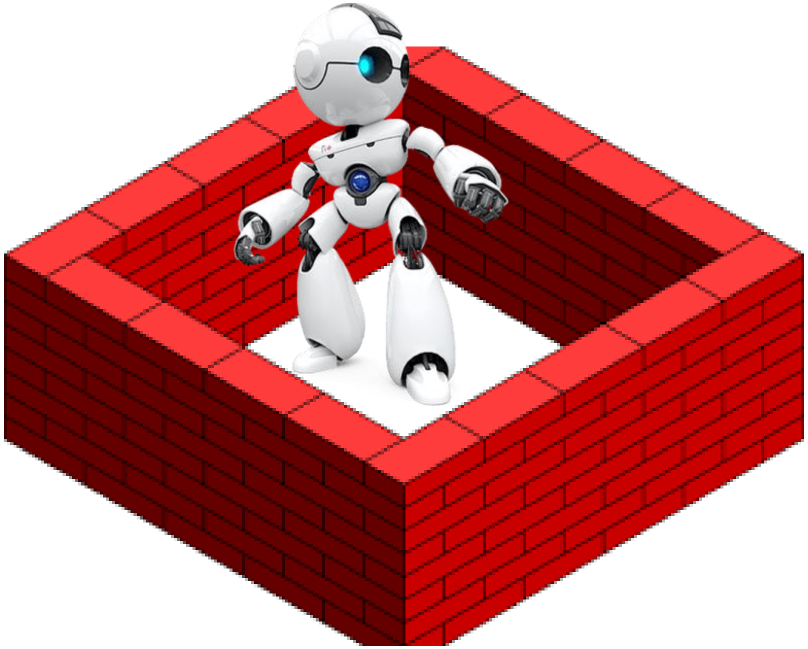
$\rightarrow$ Not practical for many large-scale problems

Quiz 02: A*

- Work out the order in which states are expanded, as well as the path returned by graph search. Assume ties resolve in such a way that states with earlier alphabetical order are expanded first.



Memory-bounded heuristic search



Memory-bound heuristic search

- In practice, A* usually runs out of space long before it runs out of time.

vs.

- **Idea:** try something like DFS, but not forget everything about the branches we have partially explored

Iterative-deepening A* (IDA*)

- The main difference with IDS
  - Cut-off use the $\boldsymbol{f}$**-value** $(\boldsymbol{g}+\boldsymbol{h})$ rather than the depth
  - At each iteration, the cutoff value is the smallest f-value of any node that exceeded the cutoff on the previous iteration
- Avoid the substantial overhead associated with keeping a sorted queue of nodes.
- Practical for many problems with unit step costs, yet difficult with real valued costs

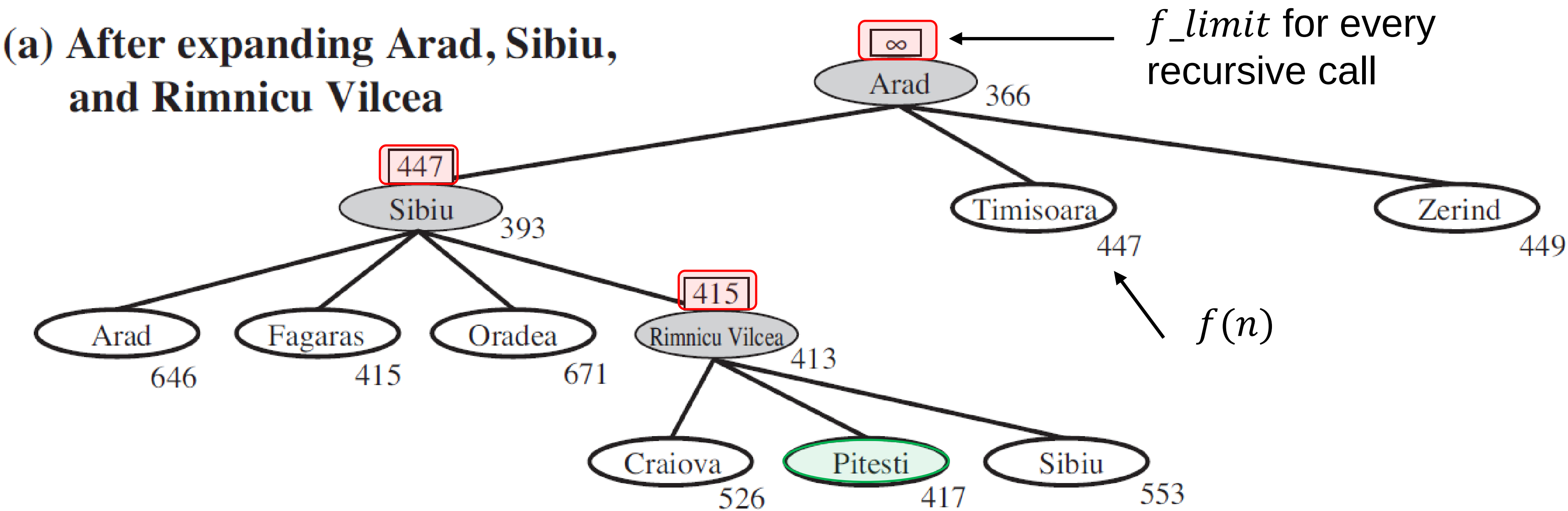
Recursive best-first search (RBFS)

- Keep track of the f-value of the best alternative path available from any ancestor of the current node

  $\rightarrow$ backtrack when the current node exceeds f_limit

- As it backtracks, replace the f-value of each node along the path with the best $f(n)$ value of its children

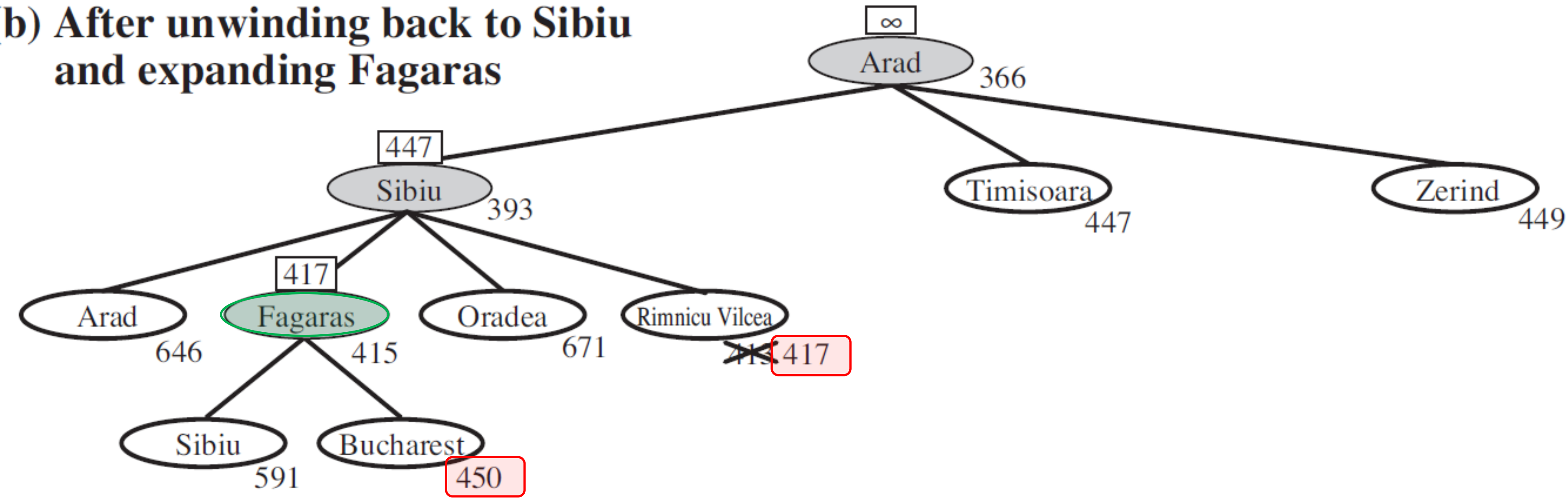
RBFS: An example



- Path until Rimnicu Vilcea is already expanded
- The path is followed until Pitesti, whose *f*-value worse than the f_limit

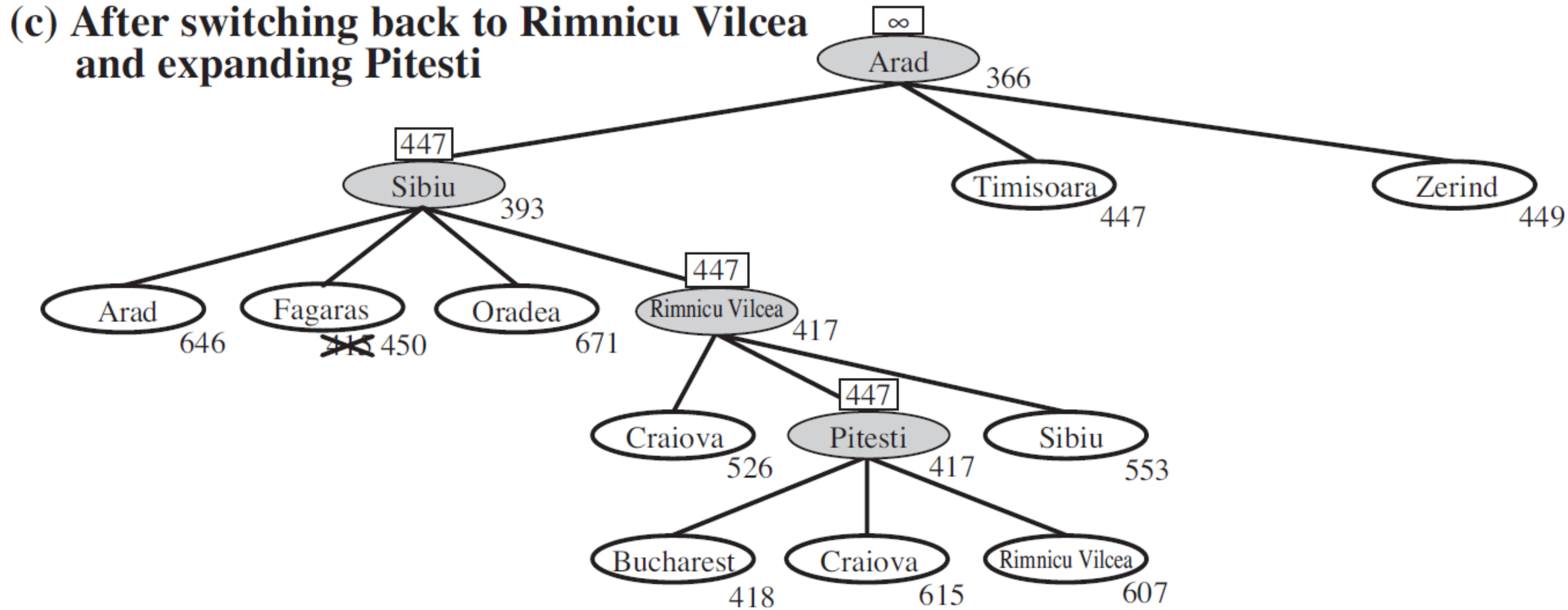
Recursive best-first search (RBFS)

(b) After unwinding back to Sibiu and expanding Fagaras



- Unwind recursion and store best f-value for current best leaf Rimnicu Vilcea
  - *result, best.f ← RBFS(problem, best, min(f_limit, alternative))*
- *best* is now Fagaras. Call RBFS for new *best*
  - *best* value is now 450

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



- Unwind recursion and store best f-value for current best leaf of Fagaras
  - *result, best.f ← RBFS(problem, best, min(f_limit, alternative))*
- *best* is now Rimnicu Viclea (again). Call RBFS for new *best*
  - Subtree is again expanded
  - Best alternative subtree is now through Timisoara
- Solution is found since because 447 > 418.

Recursive best-first search (RBFS)

function RECURSIVE-BEST-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  return RBFS(*problem*, MAKE-NODE(*problem*.INITIAL-STATE),∞)

---

function RBFS(*problem, node, f_limit*) **returns** a solution, or failure and a new *f*-cost limit
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  successors ← [ ]
  for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    add CHILD-NODE(*problem, node, action*) into *successors*
  if *successors* is empty **then return** failure, ∞
  for each *s* **in** *successors* **do** /* update *f* with value from previous search, if any */
    s.f ← max(*s.g*+*s.h*, *node.f*))
  loop do
    best ← the lowest *f*-value node in *successors*
    if *best.f* > *f_limit* **then return** failure, *best.f*
    alternative ← the second-lowest *f*-value among *successors*
    result, best.f ← RBFS(*problem*, *best*, min(*f_limit, alternative*))
    if *result* ≠ failure then return *result*

Evaluation of RBFS

- Optimality
  - Like A*, optimal if $h(n)$ is admissible
- Time complexity
  - Difficult to characterize
  - Depends on accuracy of $h(n)$ and how often best path changes
  - Can end up “switching” back and forth
- Space complexity
  - Linear time: *O(bd)*
  - Other extreme to A* - uses **too little** memory even if more memory were available

(Simplified) Memory-bound A* – (S)MA*

- Like A*, but delete the worst node (largest f-value) when memory is full
- Also backs up the value of the forgotten node to its parent
  - If there is a tie (equal f-values), delete the oldest nodes first
- Find an optimal reachable solution given memory constraint
  - The depth of the shallowest goal node is less than the memory size (expressed in nodes).
- Time can still be exponential.

Learning to search better

- *Could an agent learn how to search better?* **YES**
- **Metalevel state space:** in which each state captures the internal (computational) state of a program that is searching in an object-level state space.
- For example, the map of Romania problem,
  - The internal state of the A* algorithm is the current search tree.
  - Each action in the metalevel state space is a computation step that alters the internal state, e.g., [expands a leaf node and adds its successors to the tree]

Learning to search better

- The expansion of Fagaras is not helpful → harder problems may even include more such missteps

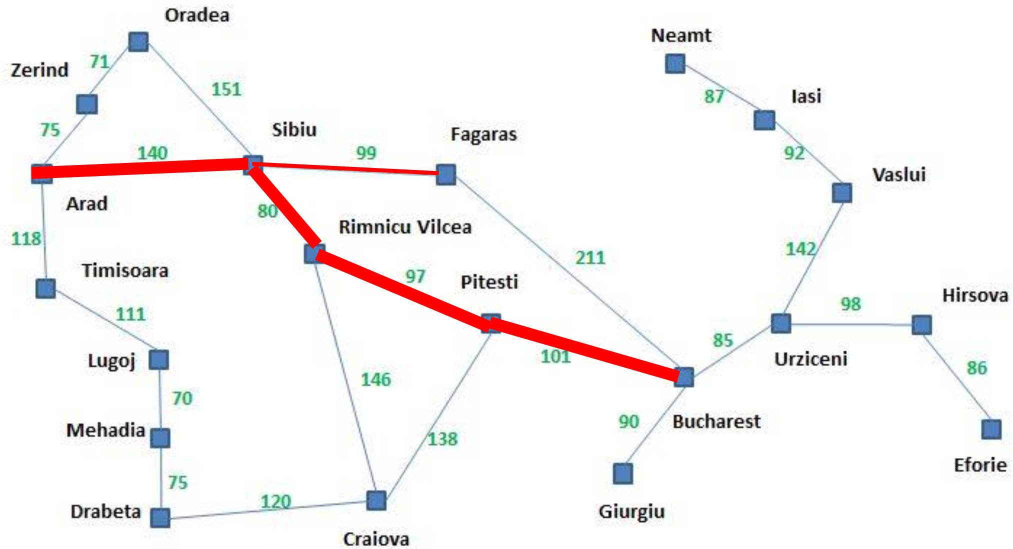


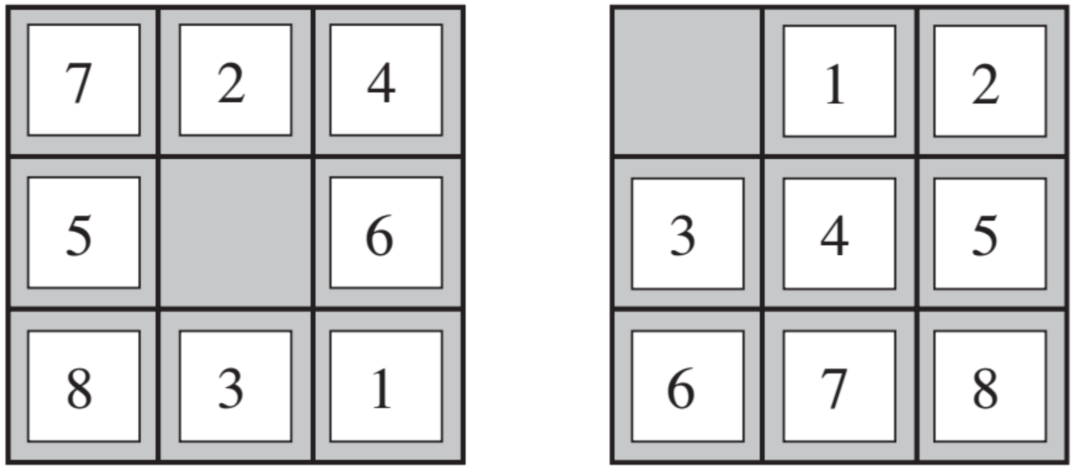
Figure A simplified road map of part of Romania.

- A metalevel learning algorithm gains from these experiences to avoid exploring unpromising subtrees

  → reinforcement learning

Heuristic functions

The 8-puzzle problem

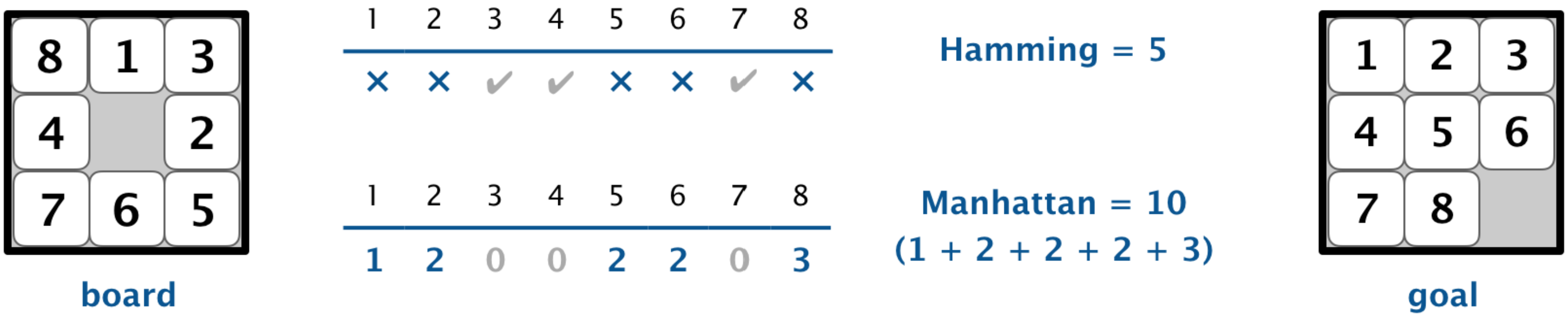


A typical instance of the 8-puzzle. The solution is 26 steps long.

- Average solution cost: about 22 steps, branching factor ~ 3.
- 8-puzzle: 9!/2 = 181,440 reachable states
- 15-puzzle: 1.05 x 10^{13} possible states

Admissible heuristics for 8-puzzle

- $h_1(n)$ = *number of misplaced numbered tiles (Hamming distance)*



- $h_2(n)$ = *sum of the (Manhattan) distance of every numbered tile to its goal position*

Quiz 03: Admissible heuristics

- Knowing that $h(n) \leq h^*(n)$
- For 8-puzzle, which of the following heuristics is admissible?

  - $h_1(n) = \text{total number of misplaced tiles}$
  - $h_2(n) = \text{total Manhattan distance}$
  - $h_3(n) = 0$
  - $h_4(n) = 1$
  - $h_5(n) = h^*(n)$
  - $h_6(n) = \min(2, h^*(n))$
  - $h_7(n) = \max(2, h^*(n))$

The effect of heuristic on performance

- Effective branching factor b^*: the factor that a uniform tree of depth d would have to contain $N+1$ nodes

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

  - where N is the total number of nodes generated by A* for a particular problem and d is the solution depth
  - E.g., A* finds a solution at depth 5 using 52 nodes $\rightarrow b^* = 1.92$

- b^* varies across problem instances, but fairly constant for sufficiently hard problems
- A well-designed heuristic would have a value of b^* close to 1

$\rightarrow$ fairly large problems solved at reasonable cost

Search cost vs. Branching factor

	Search Cost (nodes generated)			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	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	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

Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A$*$ algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

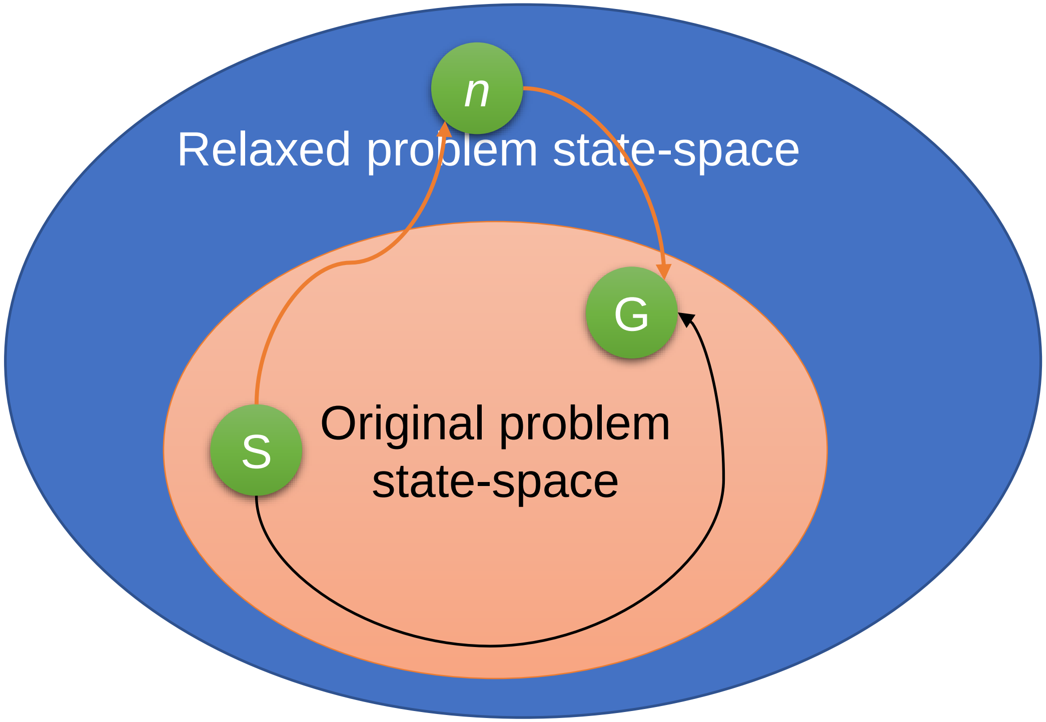
Heuristic dominance

- Given two admissible heuristics, h_1 and h_2
- If $h_2(n) \geq h_1(n)$, for all n, then h_2 **dominates** h_1
  - A* using h_2 will never expand more nodes than A* using h_1
- Better to use a heuristic function with higher values, provided it is consistent and its computation time is not too long.

How might one have come up with h_2?
Is it possible to invent such a heuristic mechanically?

Relaxed problems

- Problems with fewer restrictions on the actions



The state-space graph of the relaxed problem is a *supergraph* of the original state space

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

Relaxed problems of the 8-puzzle

- Original problem:
  - **A tile can move from square A to square B** if A is horizontally or vertically adjacent to B and B is blank
- Relaxed problems are generated by removing one or both conditions
  - A tile can move from square A to square B if A is adjacent to B.
  - A tile can move from square A to square B if B is blank.
  - A tile can move from square A to square B.

Manhattan distance

Misplaced tiles

It is crucial that the relaxed problems generated by this technique can be solved essentially *without search*

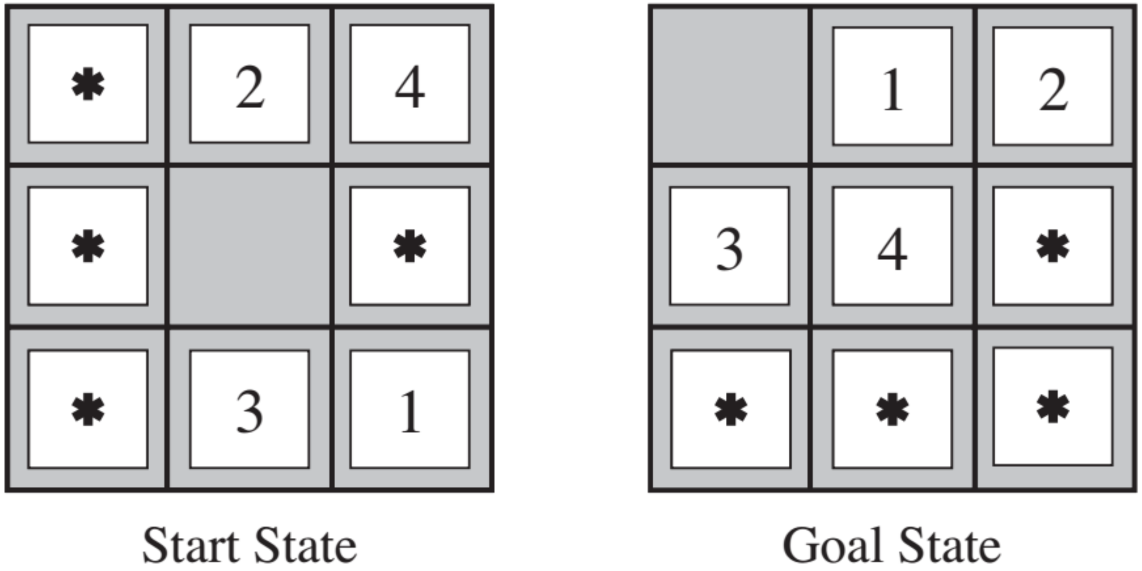
Relaxed problems

- Given a collection of admissible heuristics, $\boldsymbol{h_1}$, $\boldsymbol{h_2, \ldots,}$ $\boldsymbol{h_m}$, available for a problem and none of them dominates any of the others.
- The composite heuristic function is defined as

$$\boldsymbol{h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}}$$

- $h(n)$ is consistent and dominates all component heuristics

A subproblem of the 8-puzzle instance

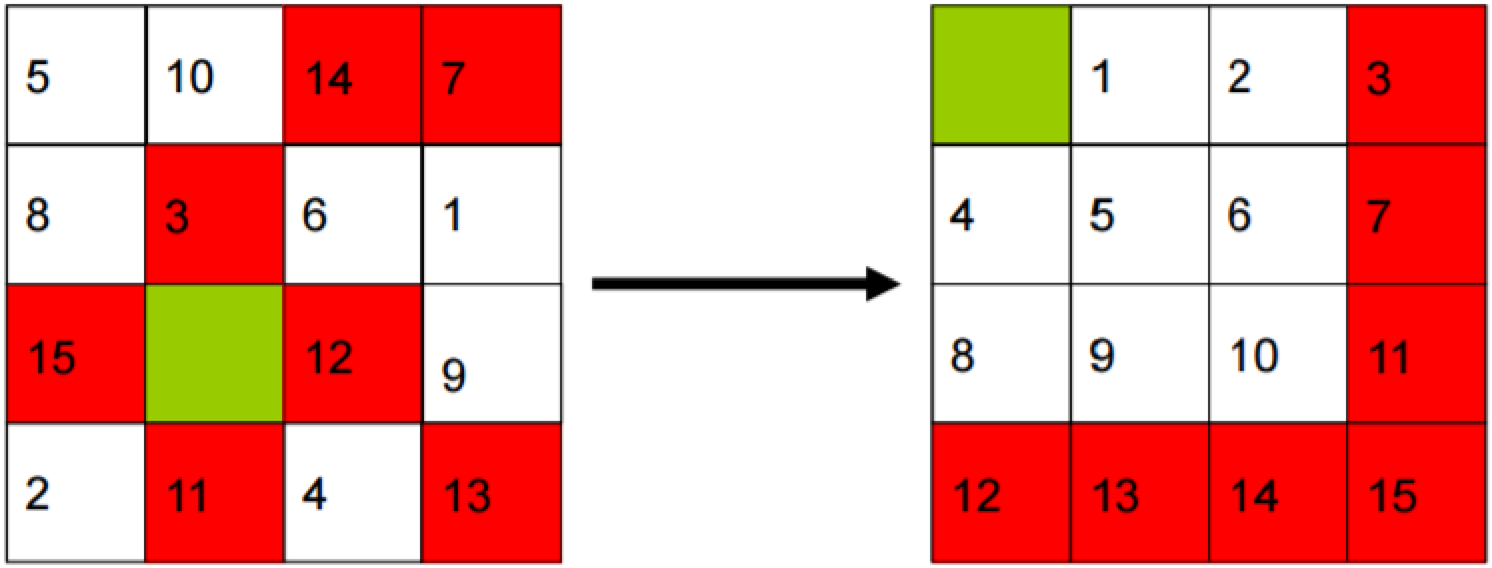


- Get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles
  - Optimal cost of this subproblem **≤** cost of the original problem
  - More accurate than Manhattan distance in some cases

Pattern databases

- Admissible heuristics can also be derived from the **solution cost** of a subproblem of the given problem.
  - This cost is a lower bound on the cost of the complete problem.
- Pattern databases (PDB): store the exact solution costs for every possible subproblem instances
  - E.g., every possible configuration of the four tiles and the blank
- The **complete heuristic** is constructed using the patterns in the databases.

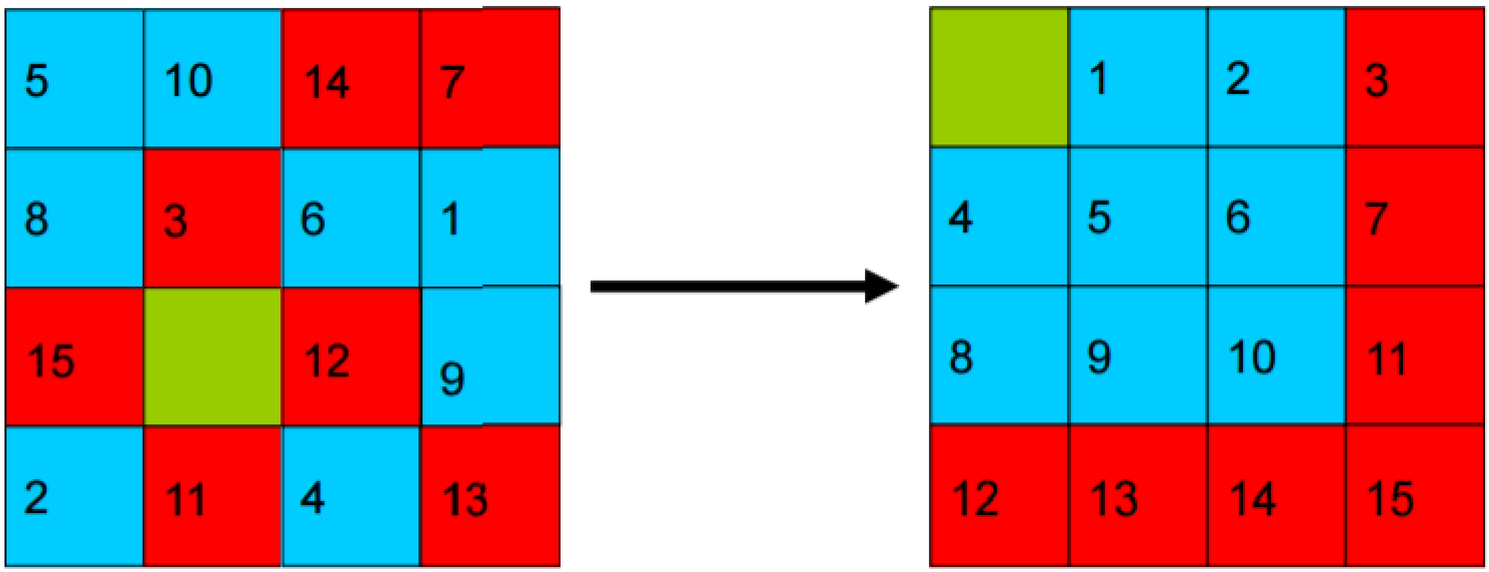
Heuristic from Pattern databases



31 moves is a lower bound on the total number of moves needed to solve this particular state

https://courses.cs.washington.edu/courses/cse473/12sp/slides/04-heuristics.pdf

Heuristic from Pattern databases



31 moves needed to solve red tiles

22 moves needed to solve blue tiles

→ Overall heuristic is maximum of 31 moves

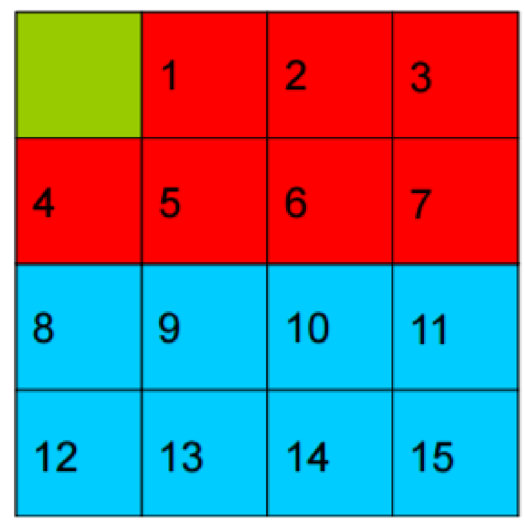
https://courses.cs.washington.edu/courses/cse473/12sp/slides/04-heuristics.pdf

Additive pattern databases

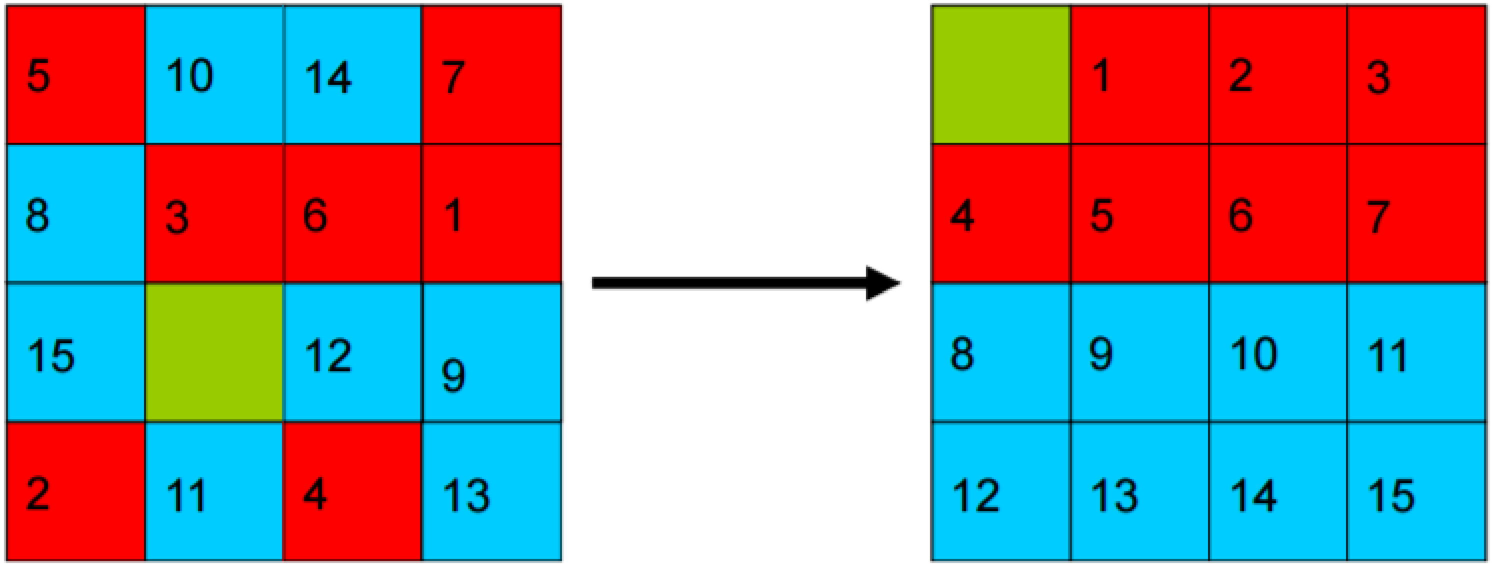
- Limitation of traditional PDB: Take max → diminish returns on additional DBs
- Disjoint pattern databases: Count only moves of the pattern tiles, ignoring non-pattern moves.
  - If no tile belongs to more than one pattern, add their heuristic values.

The 7-tile database contains 58 million entries.

The 8-tile database contains 519 million entries.



Additive pattern databases



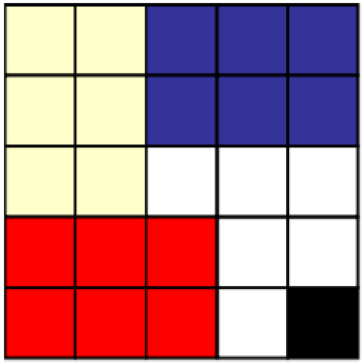
20 moves needed to solve red tiles

25 moves needed to solve blue tiles

→ Overall heuristic is 20 + 25 = 45 moves

Performance of PDB

- **15 Puzzle**
  - 2000× speedup vs. Manhattan distance
  - IDA* with the two DBs solves 15-puzzles optimally in 30 milliseconds
- **24 Puzzle**
  - 12 million × speedup vs. Manhattan
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs as shown
    - Each DB has 128 million entries
  - Without PDBs: 65,000 years



Learning heuristics from experience

- Experience means solving a lot of instances of a problem.
  - E.g., solving lots of 8-puzzles
- Each optimal solution to a problem instance provides examples from which $h(n)$ can be learned
- Learning algorithms
  - Neural nets
  - Decision trees
  - Inductive learning
  - …

THE END