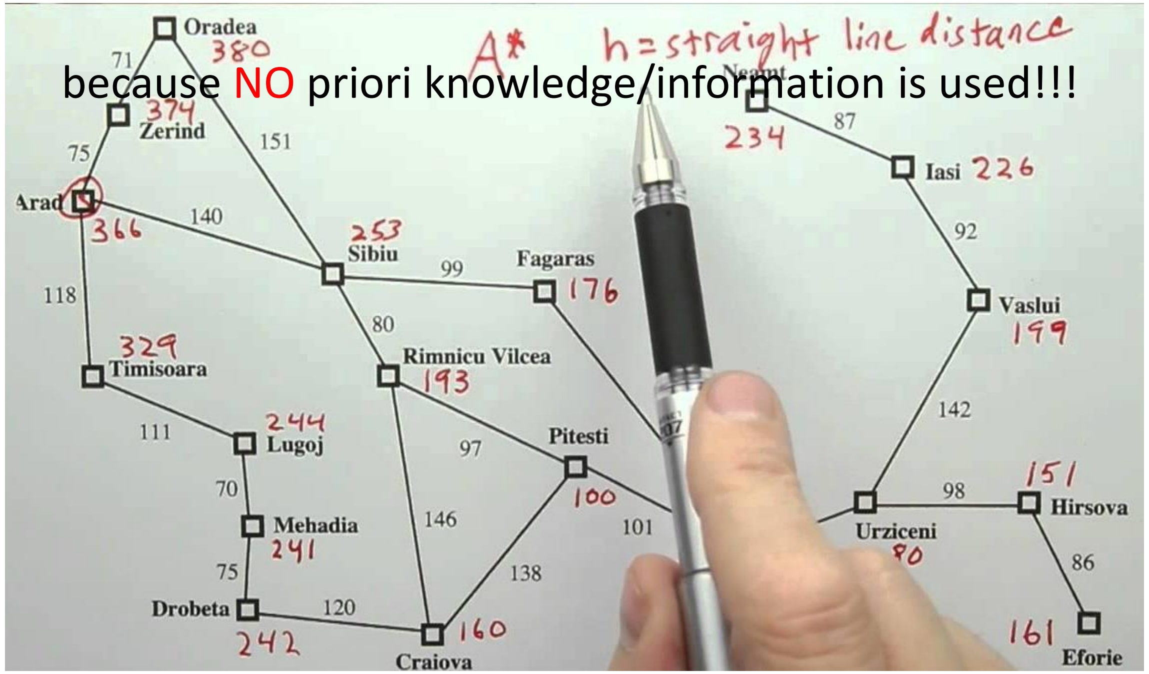
Heuristic Search

because NO priori knowledge/information is used!!!



Last week: Basic Search

- We have seen how to formalise a problem (state, action, transition)
- We have seen some basic search methods
  - Do you remember what are the methods?
  - How they differ from each other?
- Basic search methods
  - Sometimes powerful (see the reading materials of last lecture)
  - Sometimes stupid (video BFS v.s. DFS of last lecture)
- When branching factor is high…
  - Chess: 35
  - Game Go: 250

Uninformed Search -> Informed Search

- Basic search methods
  - Sometimes powerful (see the reading materials of last lecture)
  - Sometimes stupid (video BFS v.s. DFS of last lecture)
    - I want to spend less time to travel to the airport, but DFS does not care☹
    - I want to spend get a higher score in my game (collecting items before ending the game), but BFS does not understand☹
  - Why? Because NO priori knowledge/information is used!!! -> Uninformed
- Question: Can I make use of the problem-specific knowledge?

YES!!! A little AI now ☺

Outline

- Heuristic Functions
- Heuristic Search Methods
- Further Studies on Heuristics

I. Heuristic Functions

- Evaluation Function & Heuristic Function
- Admissible Heuristics

From Basic Search to Heuristic Search

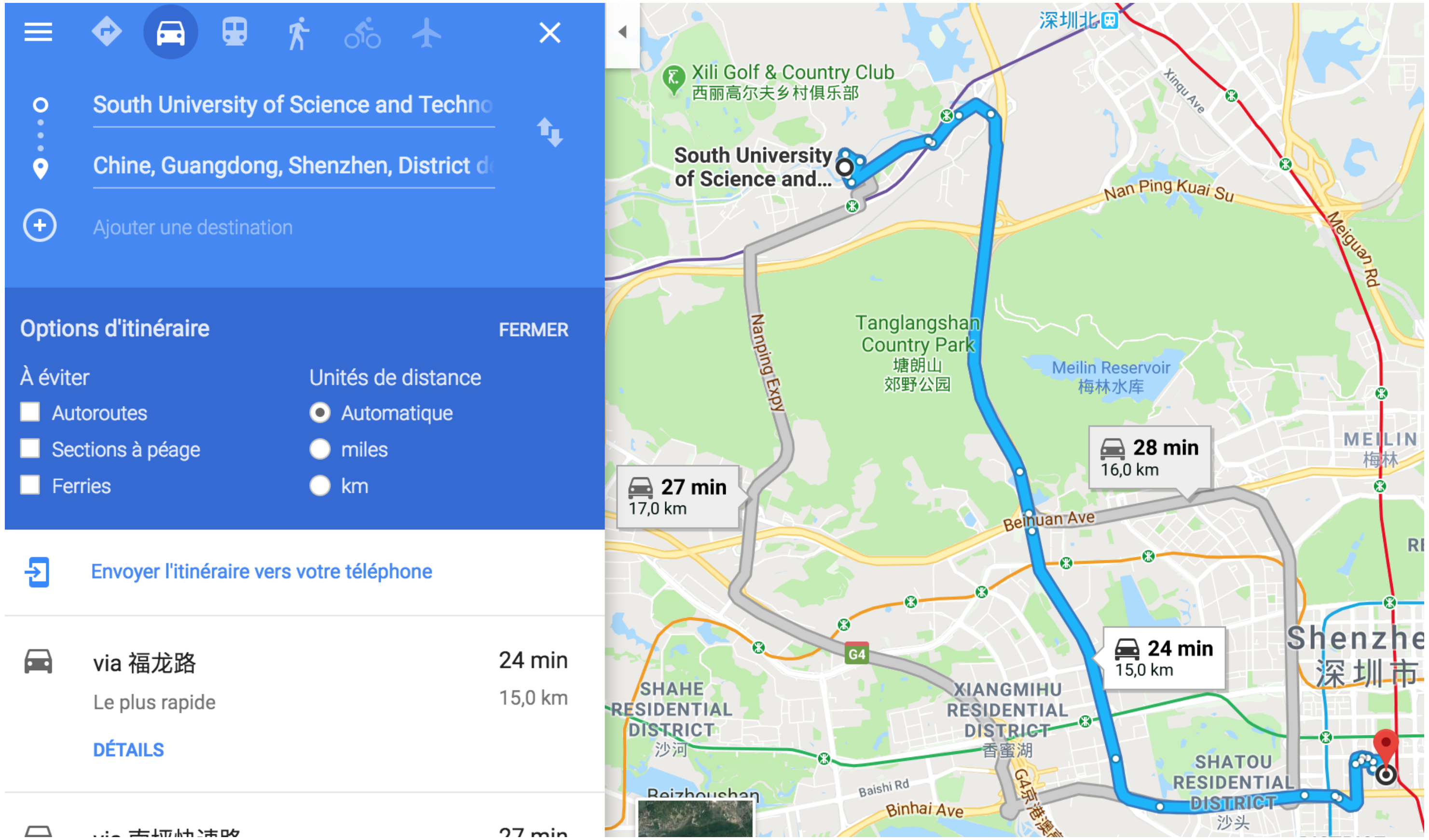
- Basic search (uninformed): use NO domain knowledge.
  - Have no bias in the search space & have to look everywhere to find the answer.
  - Time complexity is intractable for complex tasks.

- Heuristic search (informed): use domain knowledge.
  - Question: What is 'domain knowledge'?
  - Answer: Help direct/bias the search & which part of the search space to explore.
  - Can lead to drastic speed up.

Example: Route Planning

- What do you care more?
  - Faster?
  - Shorter distance?
  - Cheaper?

- Can you use some information (distance, cost, traffic, reward, etc.) to bias the search?



Evaluation Function

- Example: Best-First Search, by name, visit the "best" node first.

- To determine "best" or not, an evaluation function is needed.
  - The evaluation function, $f(s)$, estimates the cost at node s (state).
  - Several evaluation function can be designed for one single problem.
  - Best-First Search expands the node with lowest cost first.

- The meaning of cost is generalised.
  - In the route planning example, the "cost" can refer to the total distance, total time, or total expenses for traveling. *-> Decided by the customers!*
  - The "cost" can be a combination of different types of cost. People are greedy, we always want to travel faster while spending less money! *-> Trade-off!*

[Question] Difference(s) between Best-First Search and Uniform Cost Search?

[Answer] Not given right now. We will see the answer later in this lecture!

Heuristic Function

- Question: How to encode/represent domain knowledge into search?
- Answer: Heuristic function $h(s)$ at node s (state).

➢ Heuristic function $h(s)$ estimates the lowest cost from s to $Goal$.

➢ In this part, we consider a narrow case:
  - ➢ (1) $h(s) = 0$ if s is the goal node;
  - ➢ (2) nonnegative;
  - ➢ (3) problem-specific / application-dependent.

- Good h equipes search methods with some intelligence → A little AI now.

[Question] What is the difference between an evaluation function and a heuristic function?

[Answer] Heuristic is a component of an evaluation function. / Evaluation function consists of heuristic(s).

Evaluation Function and Heuristic Search

- Evaluation function $f(s)$: a cost estimate & node s with the lowest $f(s)$ is expanded first.
- Heuristic methods have $h(s)$ as a component of $f(s)$.
- The choice of f determines the search strategy.
  - Example: shortest V.S. fastest route from SUSTech to Shekou

Example: Route planning

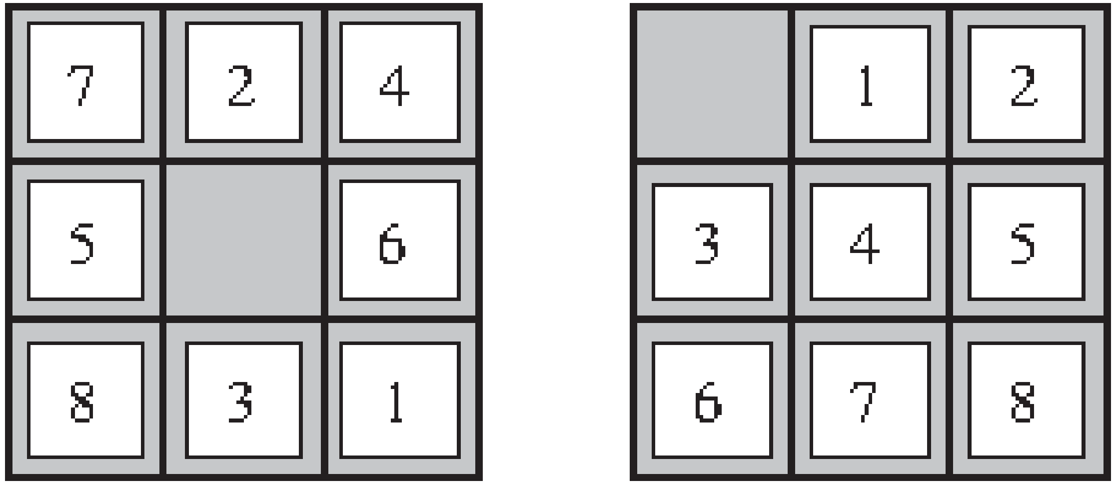
- $h_0(s) = 0$.
- $h^*(s) =$ the true cost from s to the goal.
- $h_{SLD}(s) =$ straight-line distance from node s to the goal.
- $h_{SLD}(s) + 20\%$: usually better than $h_{SLD}(s)$.

➢Find a ‘good’ h: problem-specific & a serious research problem.

Example: 8-puzzle

- Possible heuristics:
  - $h\_1(s) =$ the number of misplaced tiles
  - $h\_2(s) =$ the sum of the distances of the tiles from their goal positions -> also called *city block distance* or *Manhattan distance*

- [Question] Can you tell the
  - $h\_1(StartState)$ -> 8
  - $h\_2(StartState)$ -> 18
  - and true solution cost? -> 26



Start State

Goal State

‘Good’ Heuristics

- A heuristic can be powerful only if it is of a ‘good’ quality.
- A ‘good’ heuristic should be admissible.

Admissible Heuristics

- Admissible heuristic: $h(s)$ never overestimates the cheapest (optimal) cost from s to the goal:
  - $\forall s \rightarrow h(s) \leq h^*(s)$, where $h^*(s)$ is the true cost from s to the goal.

- Admissible heuristics are optimistic.

Admissible Heuristics: Examples

Two extreme cases:

1. The trivial $h_0(s) = 0$: No help for searching.
2. The perfect $h^*(s) =$ the true cost from s to the goal: lead directly to the best path, but unknown in practice.

Admissible Heuristics: Examples

For route planning:

- $h_{SLD}(s)$: Admissible, since a straight line is the shortest distance between two points.
- $h_{SLD}(s) + 20\%$: Not always admissible, since some may surpass the true distance to the goal.

Admissible Heuristics: Examples

For 8-puzzle:

- $h_{mis}(s)$ = #misplaced tiles $\in [0,8]$: Admissible.
- $h_{1stp}(s)$ = #(1-step move) to reach the goal: Admissible.

➢ Fact: $h_{1stp}(s) \geq h_{mis}(s)$

Exercise 1:

$h_{mis}(s) =?$

$h_{1stp}(s) =?$

7	2	4
5		6
8	3	1

➡

	1	2
3	4	5
6	7	8

- Question: which is 'better'?
- Guess: $h_{1stp}(s)$ is 'better'.
- But: what does 'better' point to? and Why?

Recap: Heuristic Search and Heuristics

- Basic search uses no domain knowledge.
- Heuristic search uses domain knowledge by a heuristic function.

- Good heuristics
    - can drastically reduce search cost;
    - should be admissible.

II. Heuristic Search Methods

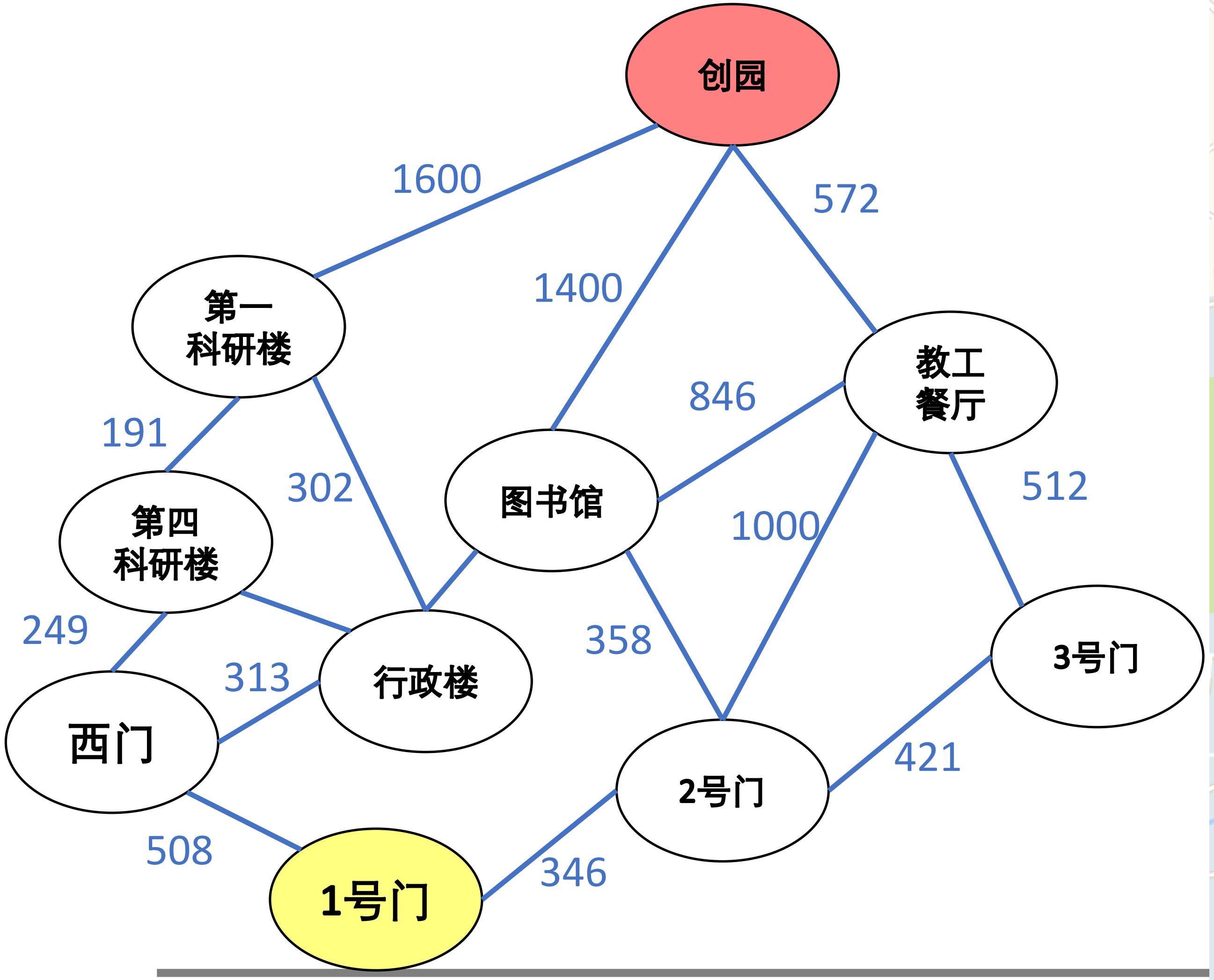
- Greedy Best-First Search
- A* Search

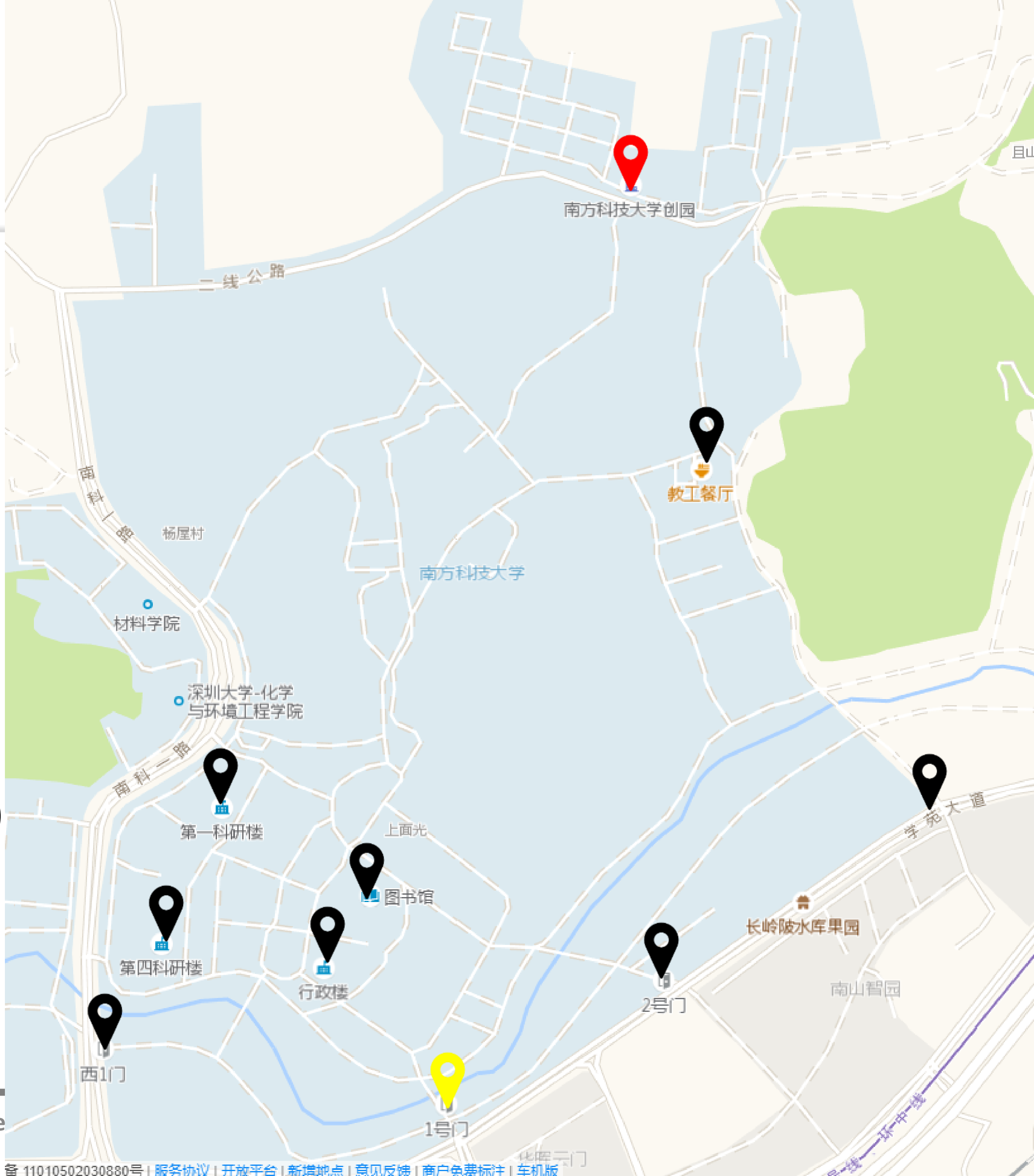
Greedy Best-first Search

Core Idea

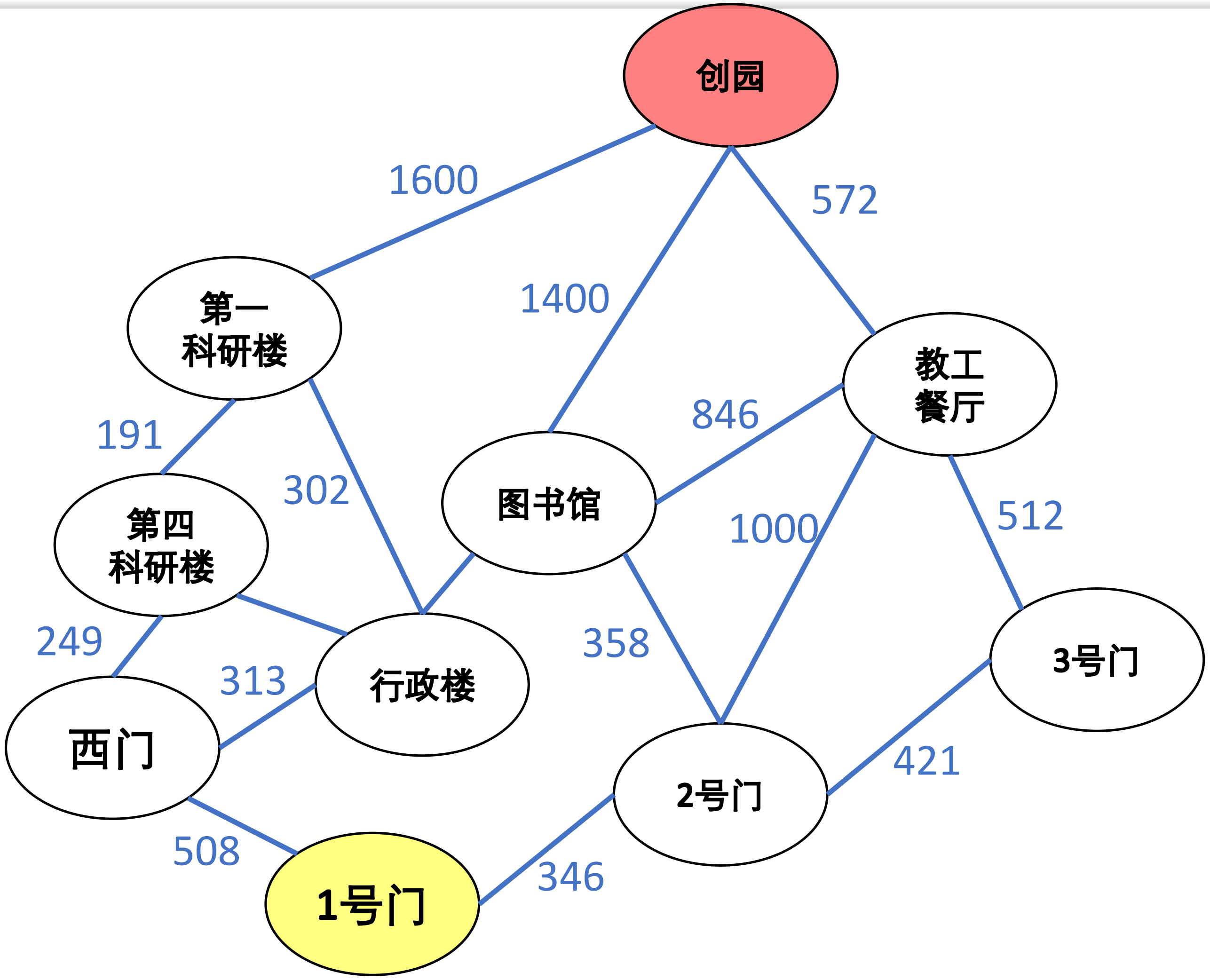
- Idea: Expand the node that seems closest to the Goal.
- Expand node s that has the minimal $f(s) = h(s)$.
- Recall: what guide the search order of uniform-cost search?

Greedy Search: Illustration 1





Greedy Search: Illustration 1



Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
二号门	1000
三号门	888

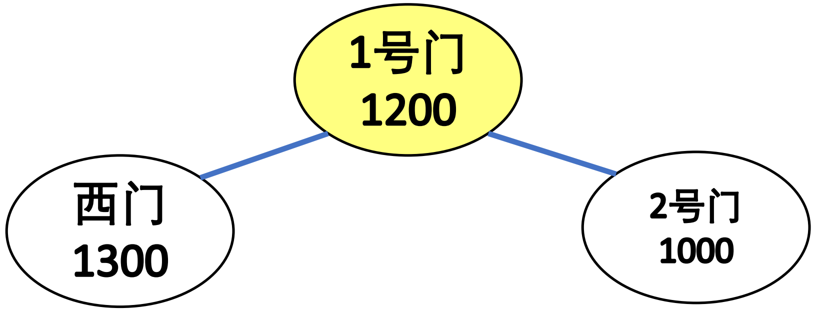
Heuristics: domain-knowledge

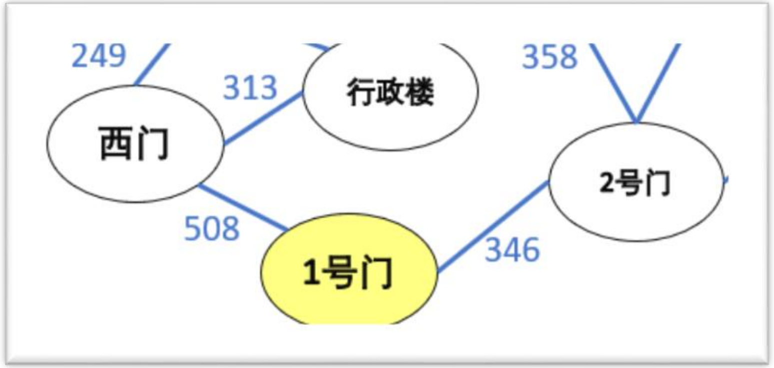
Greedy Search: Illustration 1

Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

Greedy Search: Illustration 1

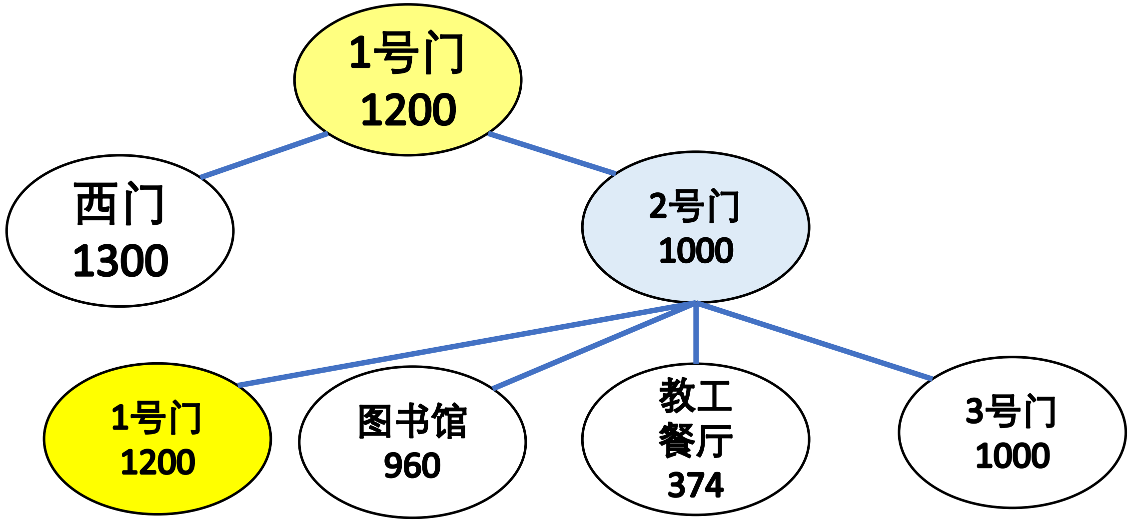
Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

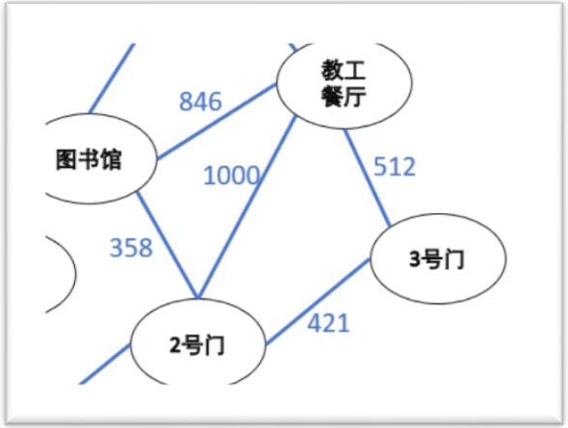




Greedy Search: Illustration 1

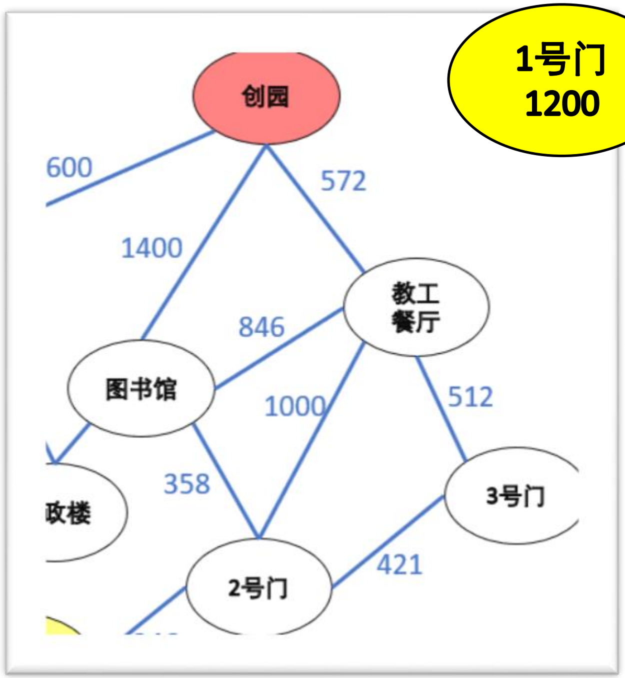
Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

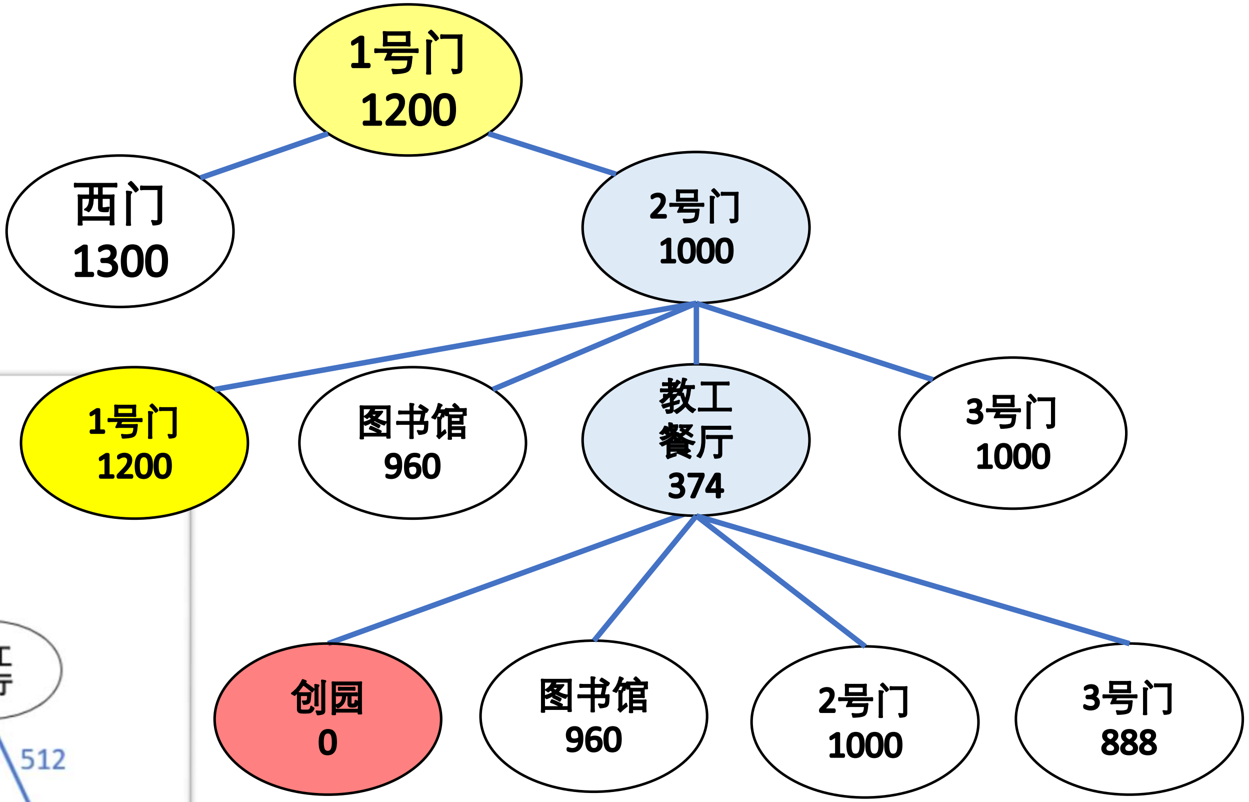




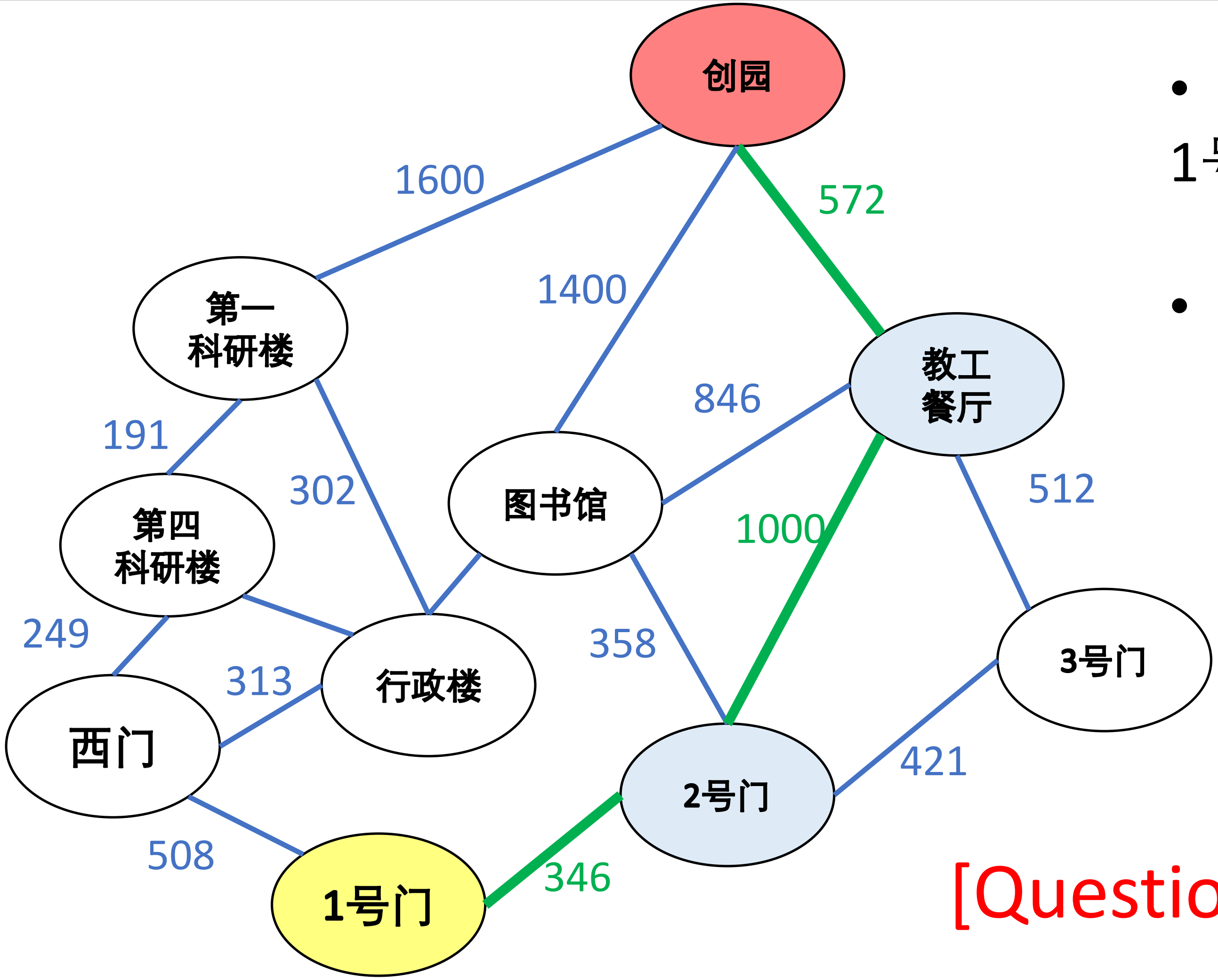
Greedy Search: Illustration 1

Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888





Greedy Search: Illustration 1

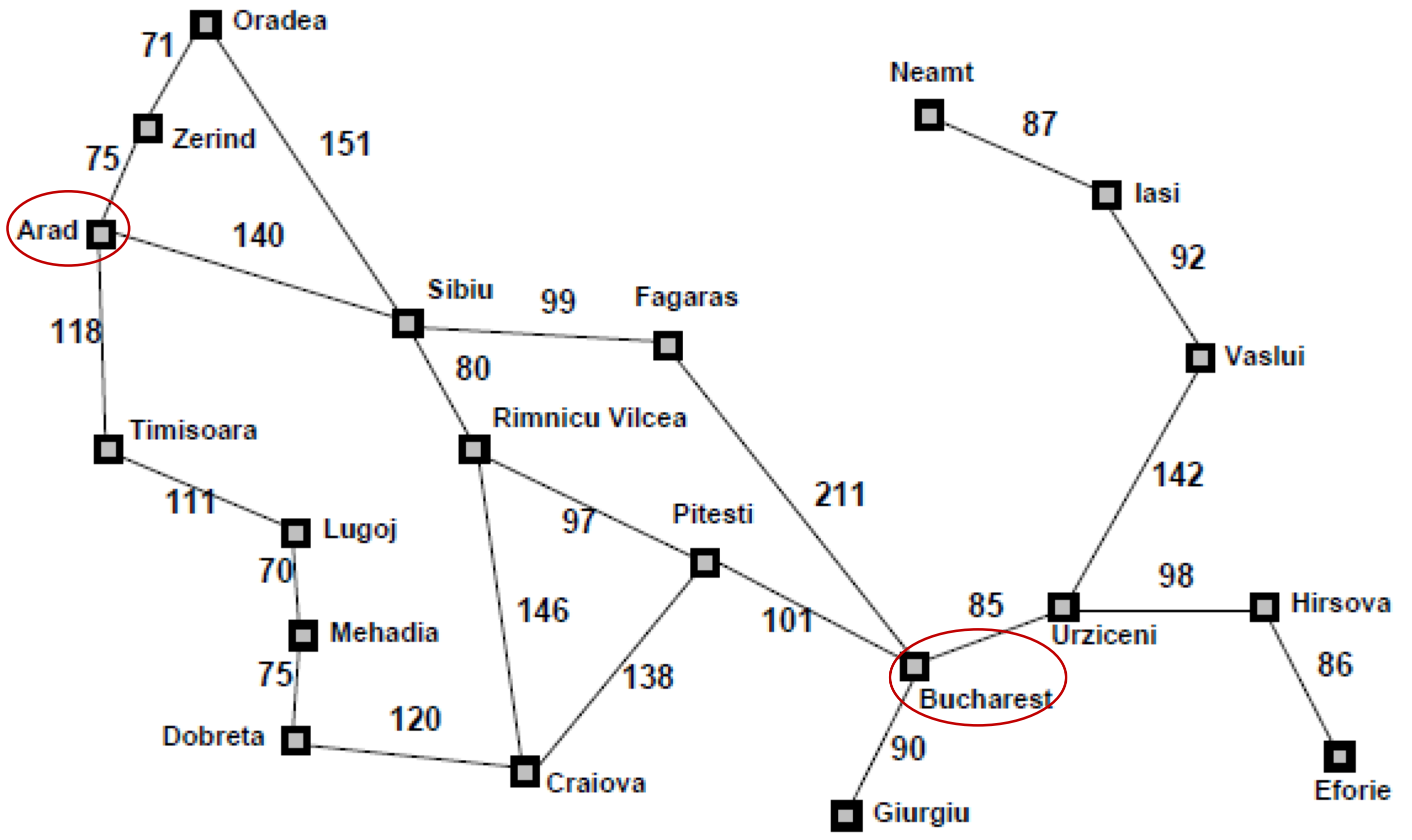


- **Solution**:
1号门->2号门 ->教工餐厅->创园

- $\boldsymbol{Distance} = 346 + 1000 + 572 = 1918m$

[Question] Is it an optimal solution?

Greedy Search: Illustration 2

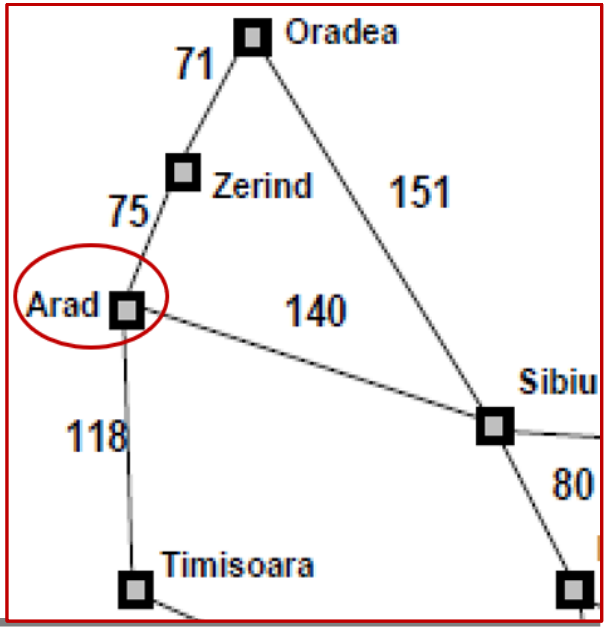


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Heuristics: domain-knowledge

Greedy Search: Illustration 2

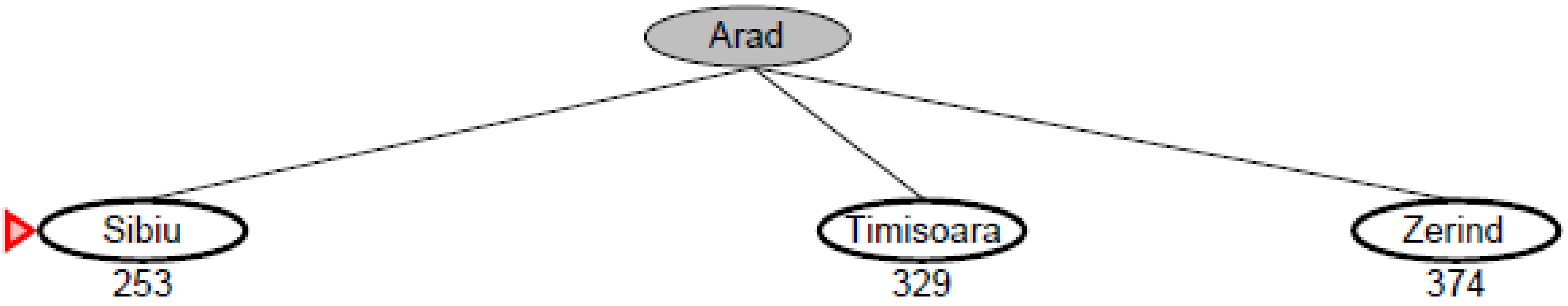
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

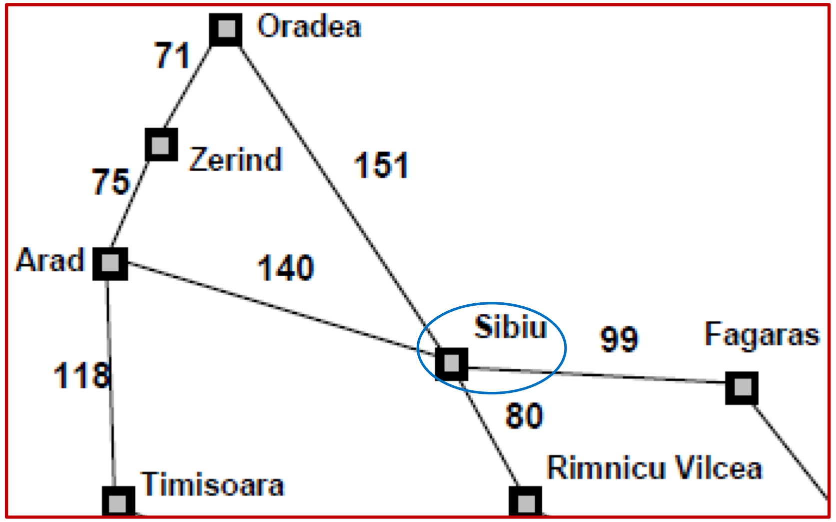


Greedy Search: Illustration 2

Straight−line distance to Bucharest

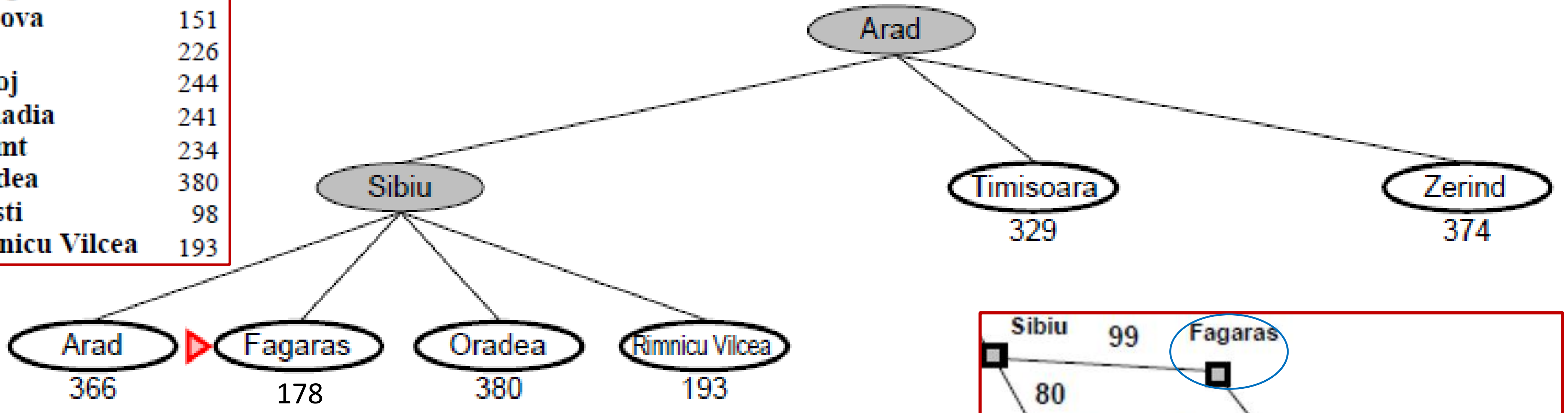
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

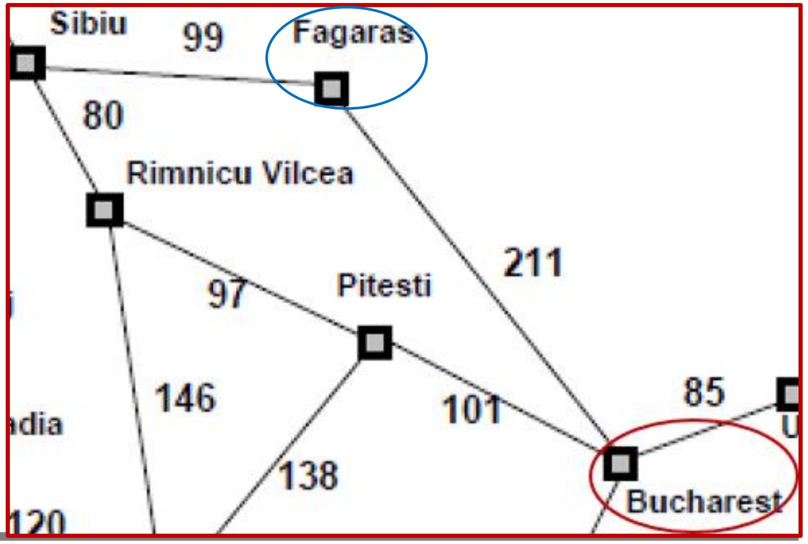




Greedy Search: Illustration 2

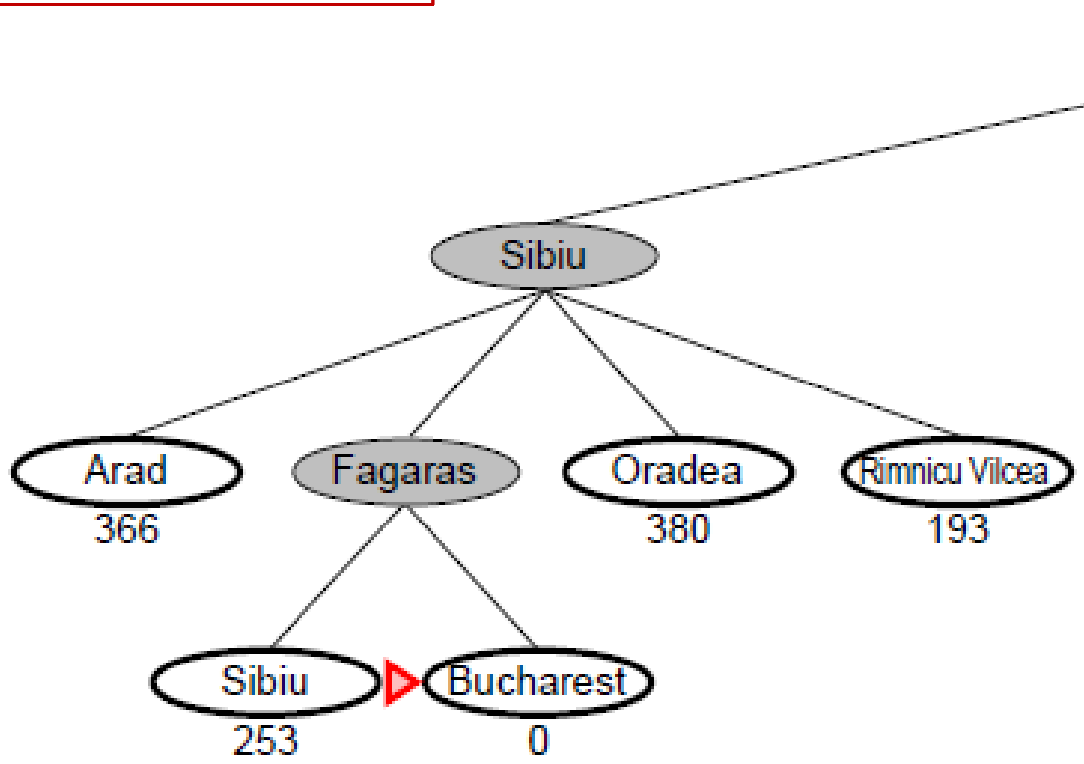
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193

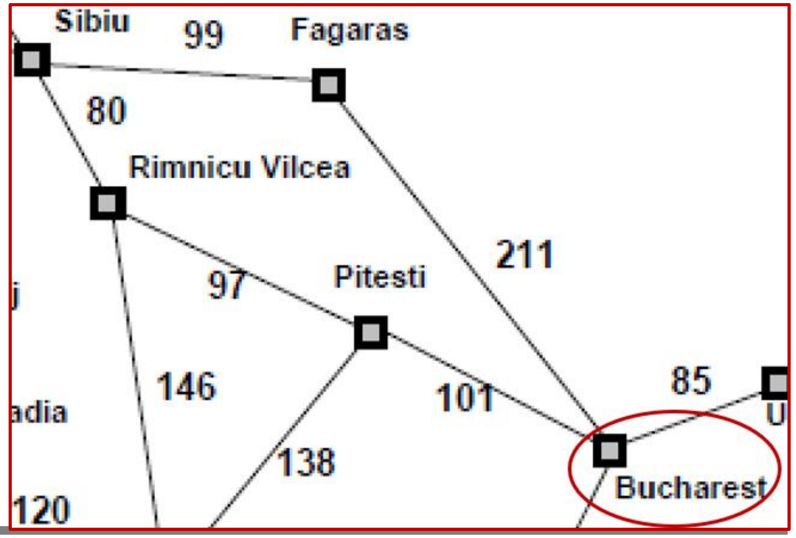




Greedy Search: Illustration 2

Sibiu	253





Core Idea Revisit

- Idea: Expand the node that seems closest to the Goal.
- Expand node s that has the minimal $f(s) = h(s)$.

Greedy Search: Performance Metrics

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

- Complete? No, can stuck in loops.
  - E.g. Oradea as the goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow \cdots$
  - Complete in the finite space with repeated-state checking.
- Optimal? No.
- Time? $O(b^m)$, but good heuristics can give drastic improvement.
- Space? $O(b^m)$, keep all nodes in memory.

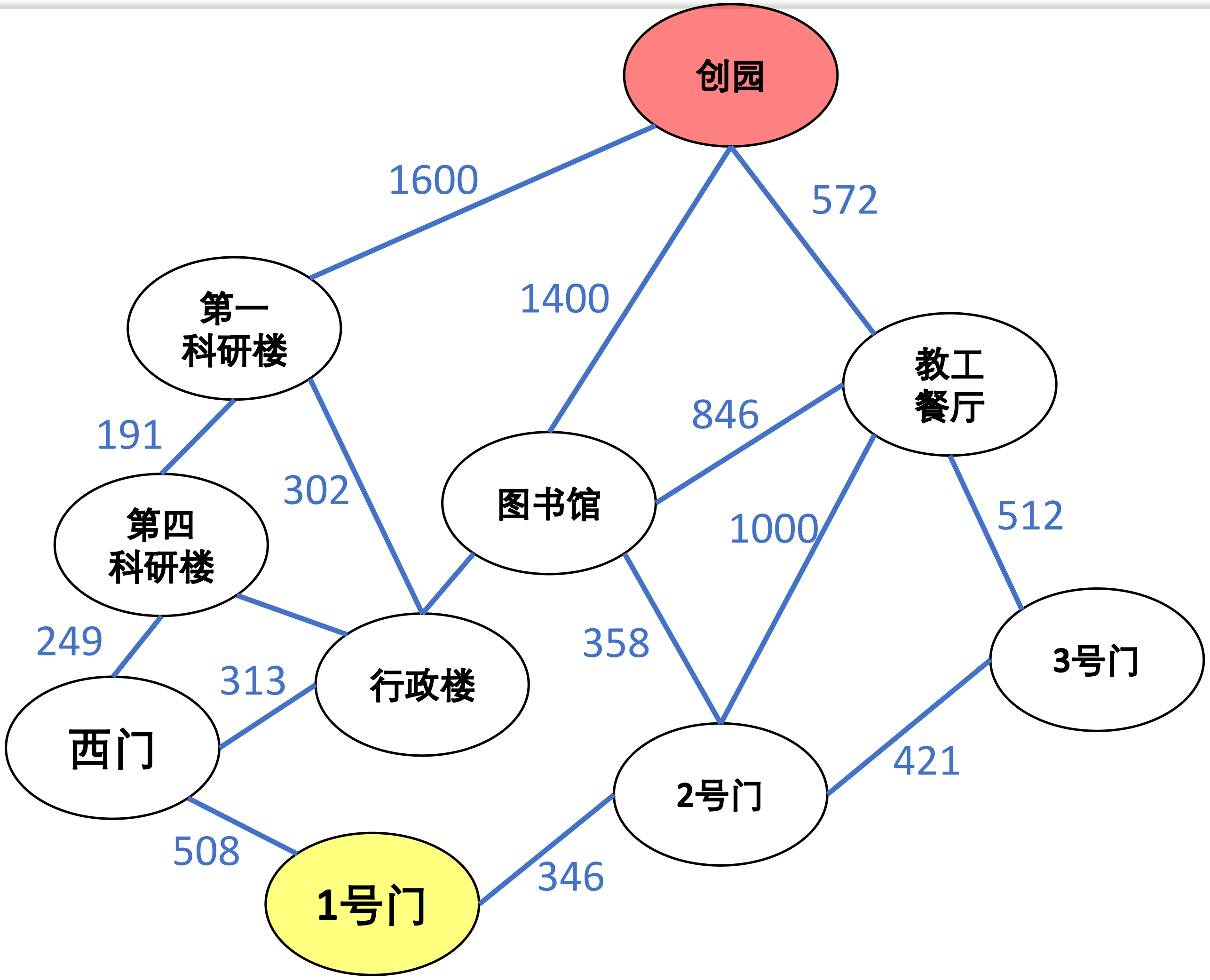
➢ Memory requirement is the biggest handicaps.

A* Search

Core Idea

- Idea: avoid expanding paths that are already expensive.
- Expand the node s that has the minimal $f(s) = h(s) + g(s)$
  - $g(s)$: cost from *Start* to s.
  - $h(s)$: estimated cost from s to *Goal*.
  - ➢ $f(s)$: estimated total cost of path from *Start* through s to *Goal*.
- Recall: what guide the search order of uniform-cost search?

A*: Illustration 1



Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
二号门	1000
三号门	888

Heuristics: domain-knowledge

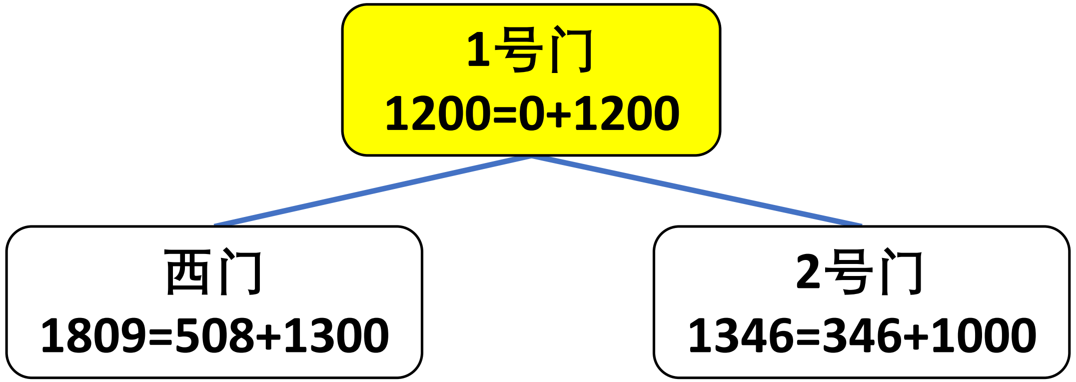
A*: Illustration 1

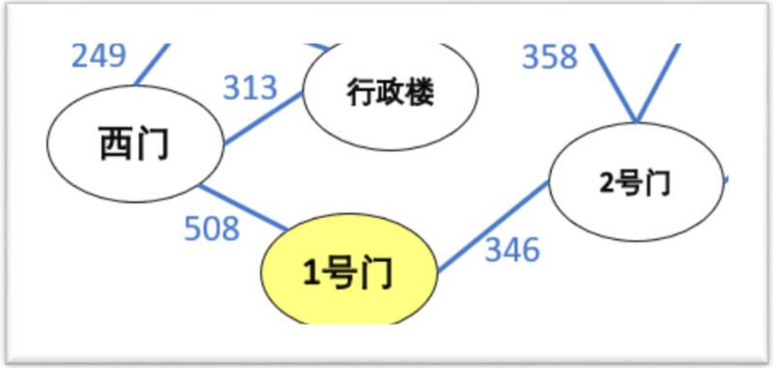
Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

1号门
1200=0+1200

A*: Illustration 1

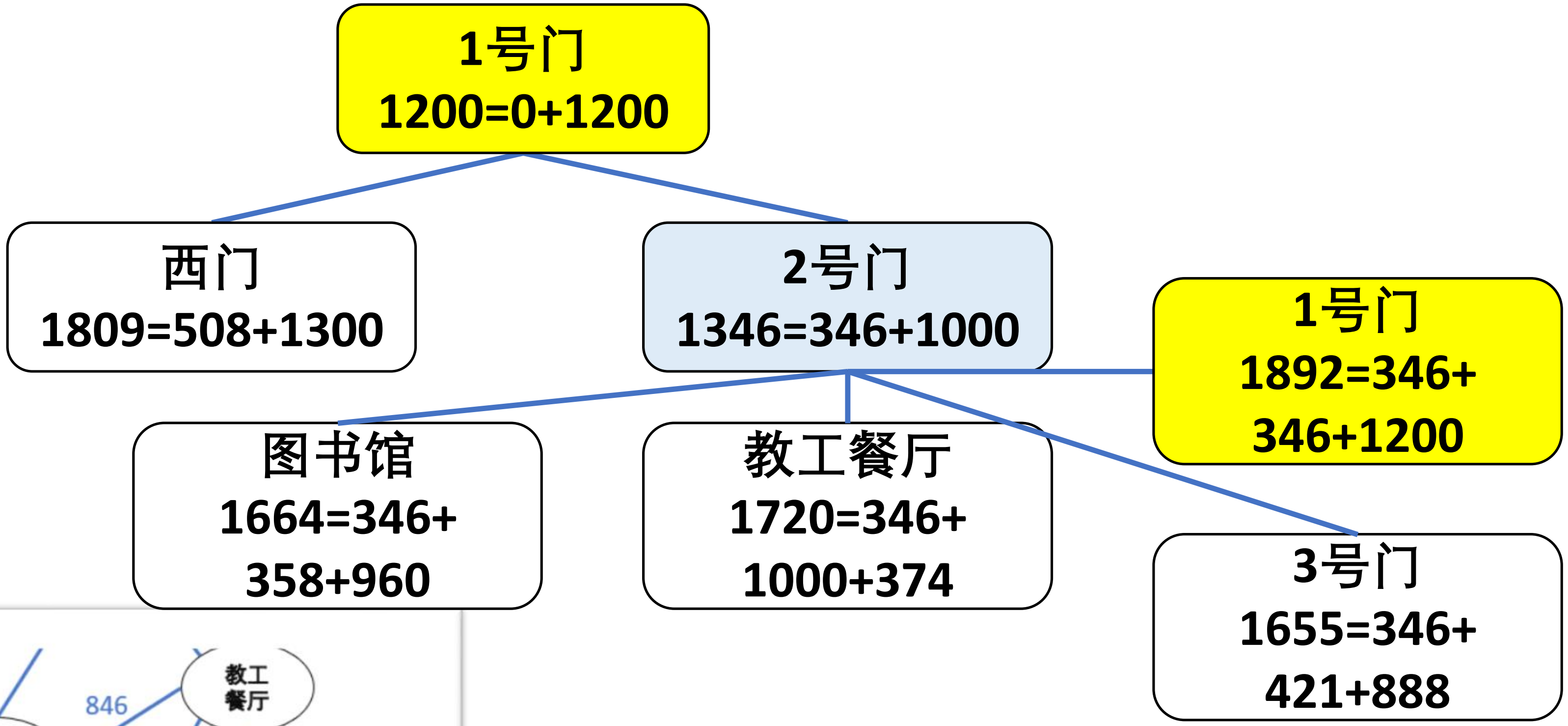
Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

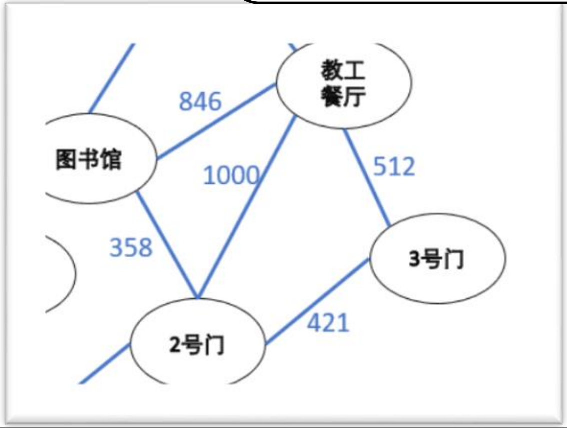




A*: Illustration 1

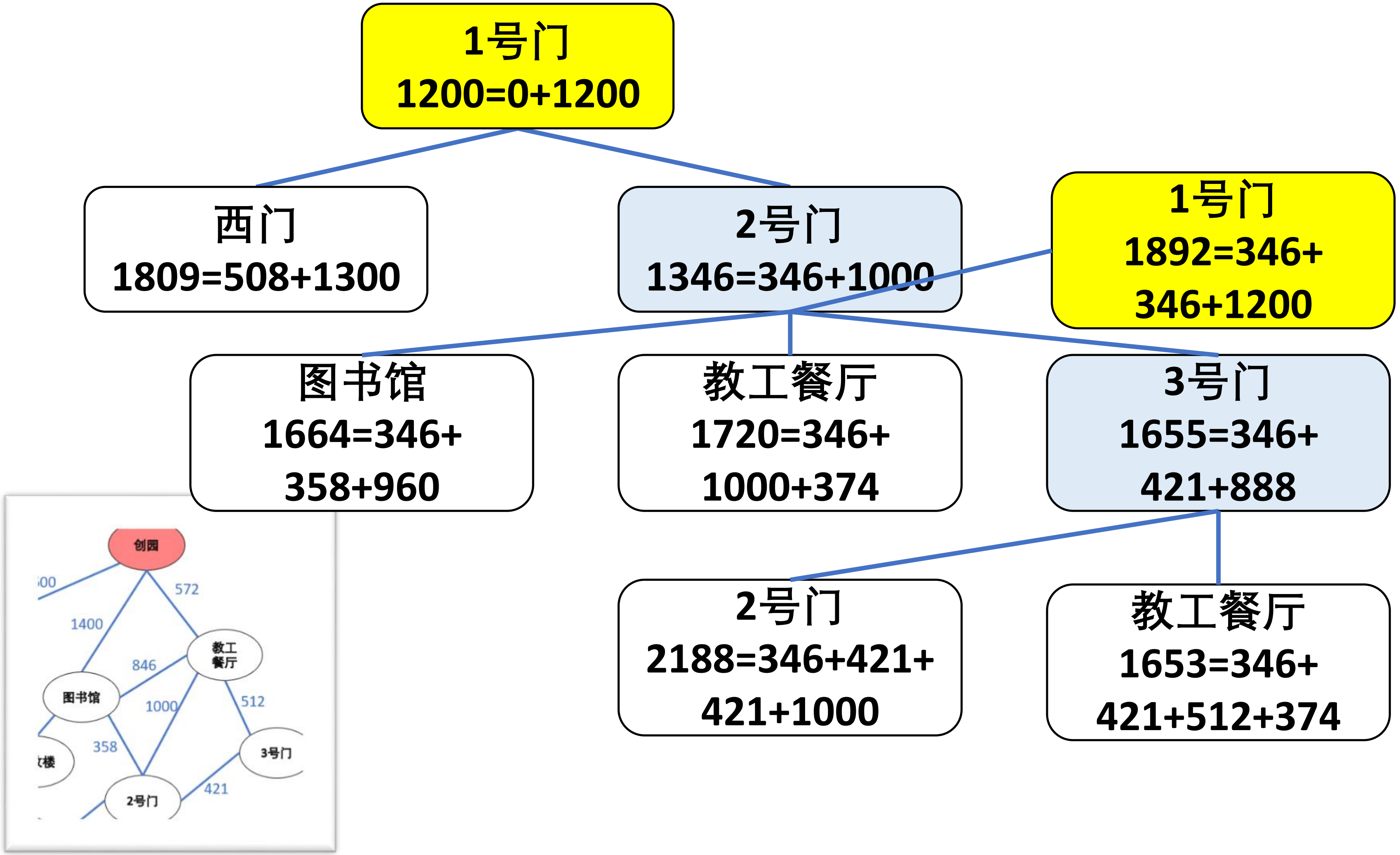
Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888





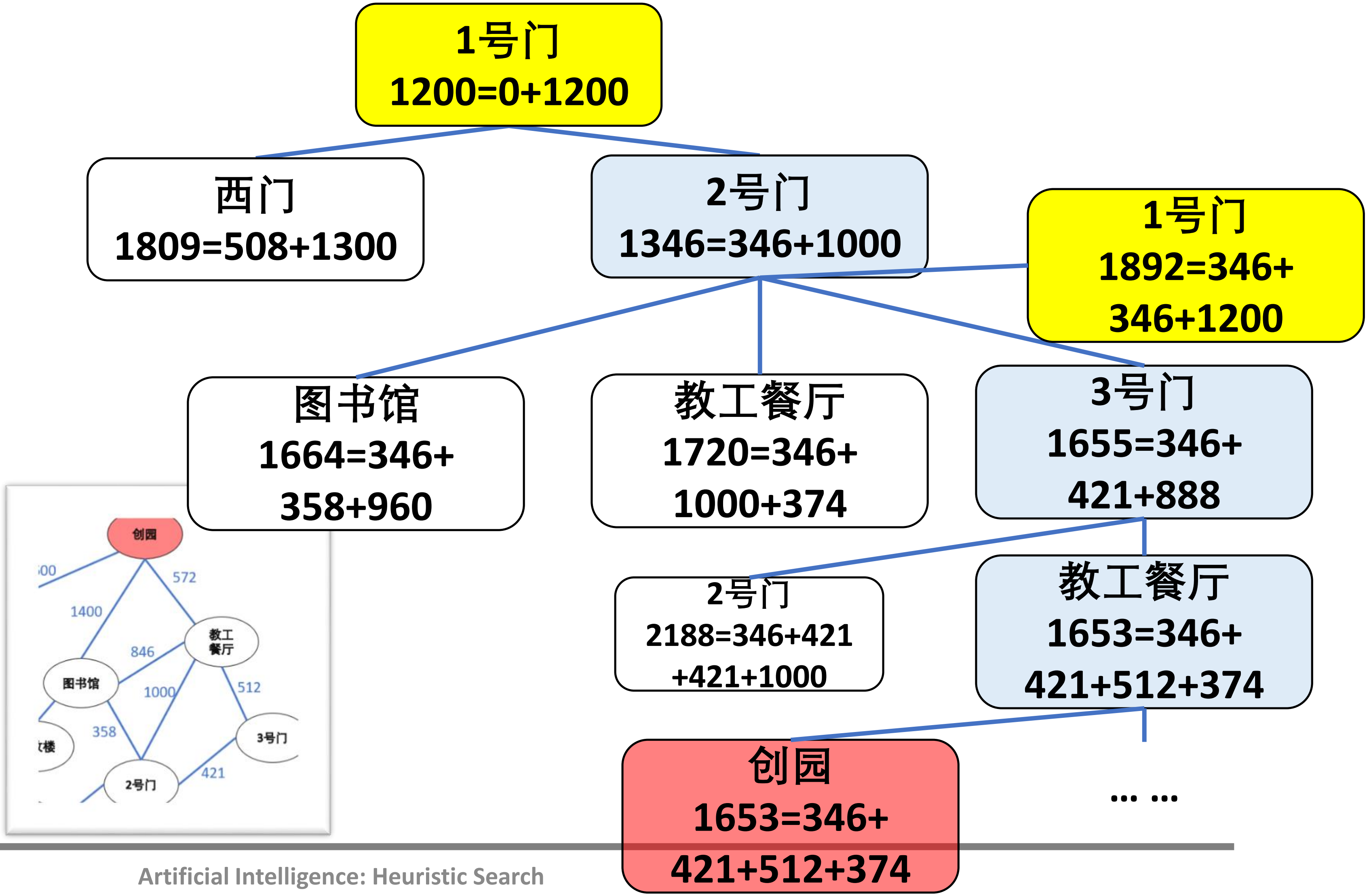
A*: Illustration 1

Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888

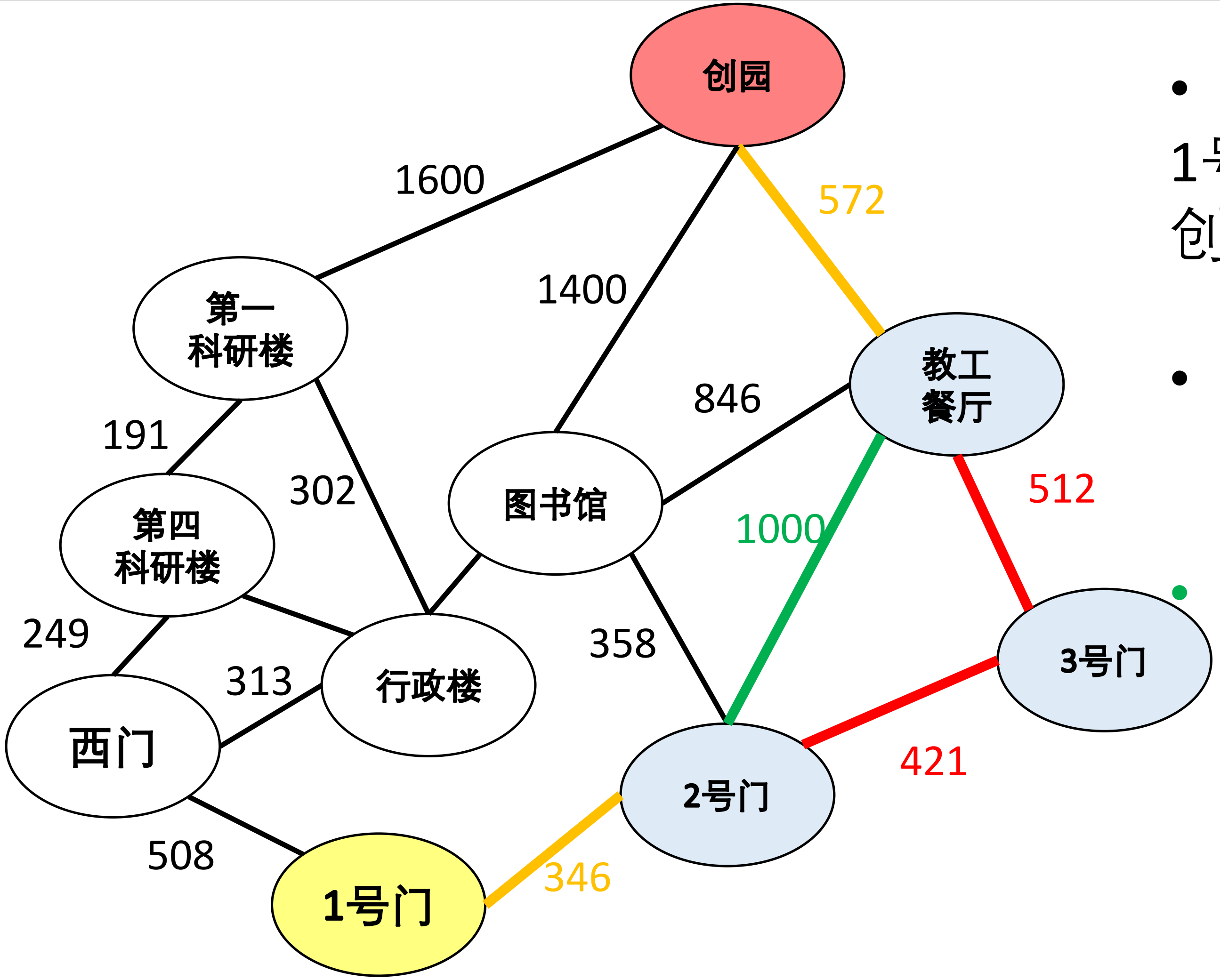


A*: Illustration 1

Node	SLD to 创园
创园	0
第一科研楼	962
第四科研楼	1080
西门	1300
1号门	1200
图书馆	960
行政楼	1100
教工餐厅	374
2号门	1000
3号门	888



A*: Illustration 1

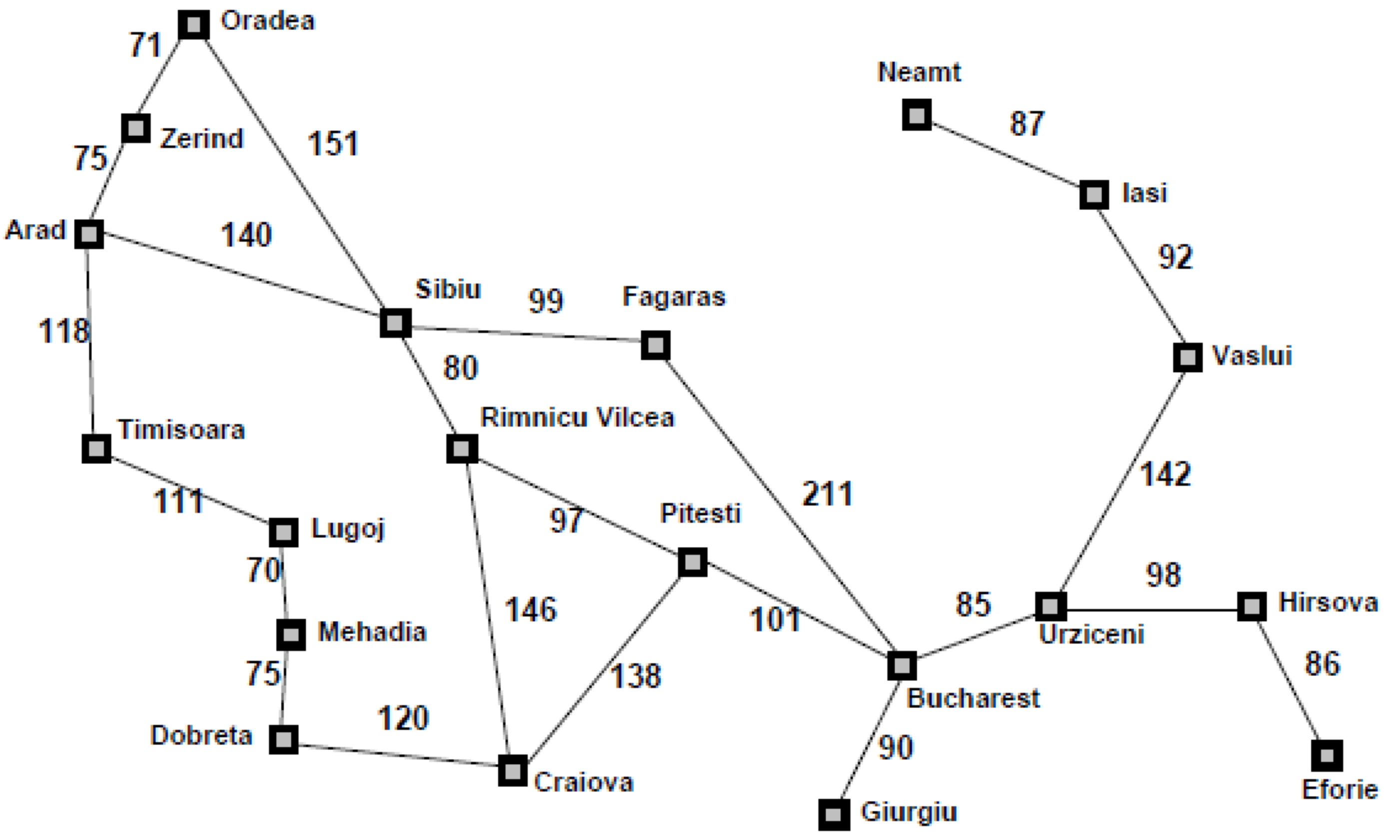


- **Solution**:
1号门->2号门->3号门->教工餐厅->创园

- $\boldsymbol{Distance} = 346 + 421 + 512 + 572 = 1851m$

- Solution by Greedy Search: 1918m

A*: Illustration 2



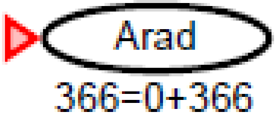
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

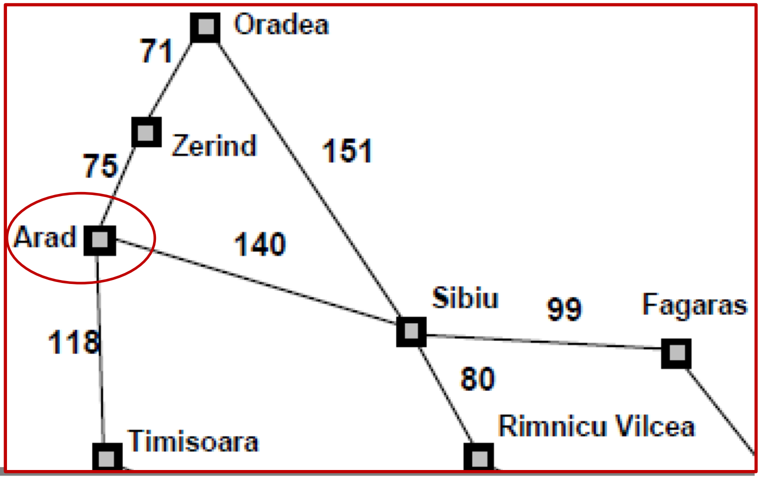
Heuristics: domain-knowledge

A*: Illustration 2

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

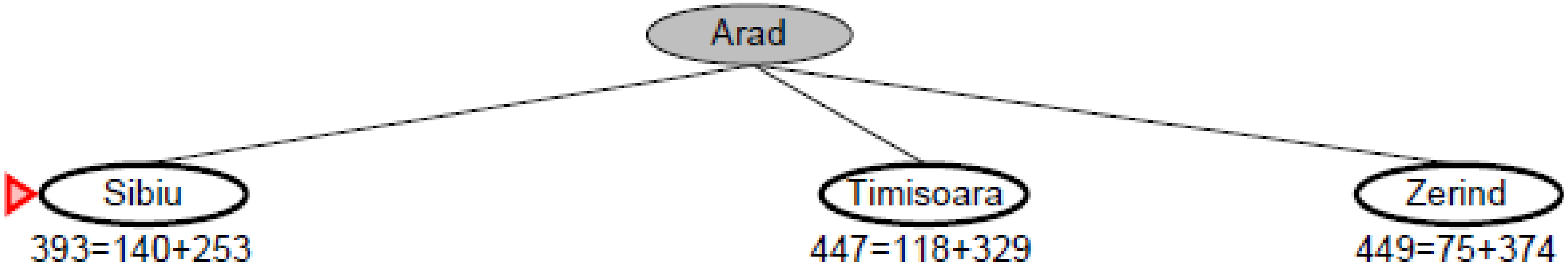


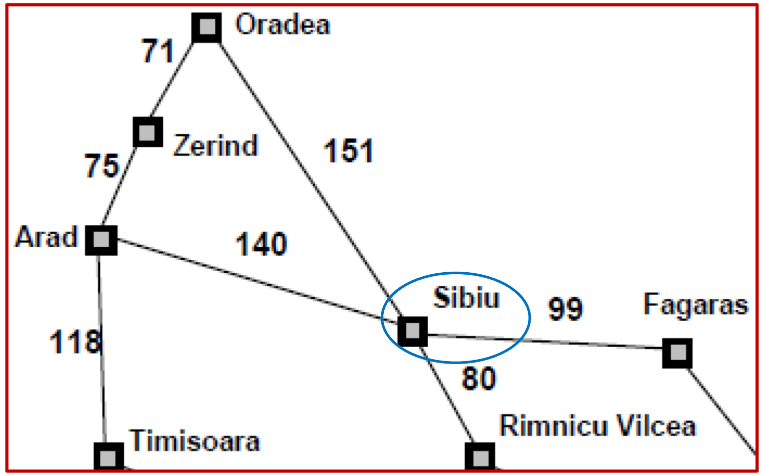


A*: Illustration 2

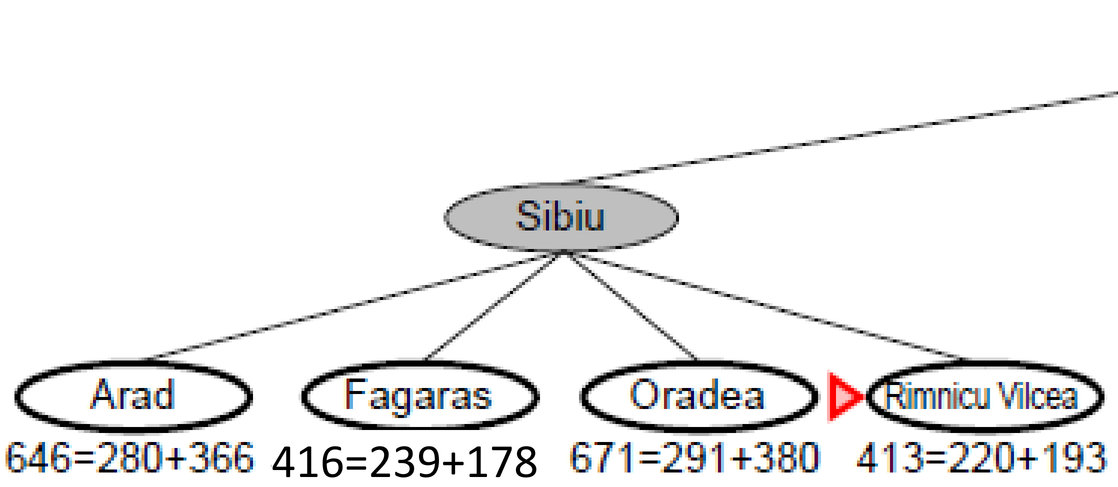
Straight−line distance to Bucharest

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

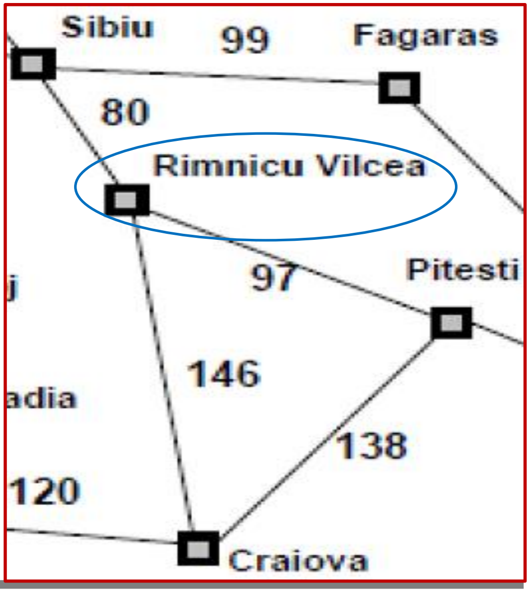




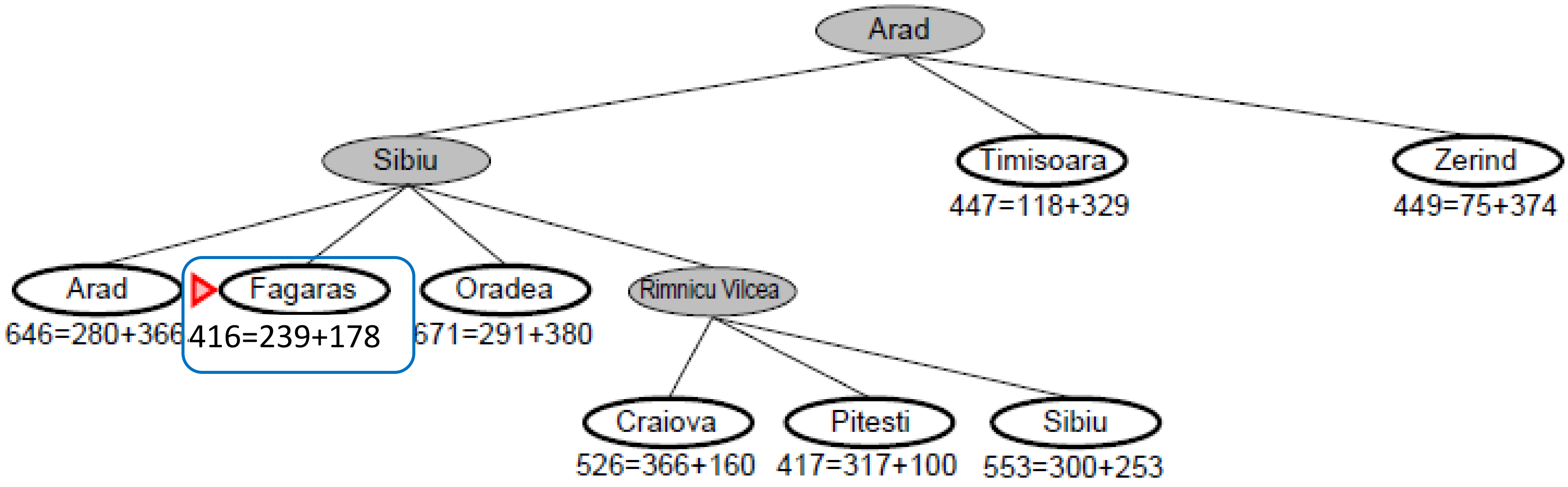
A*: Illustration 2



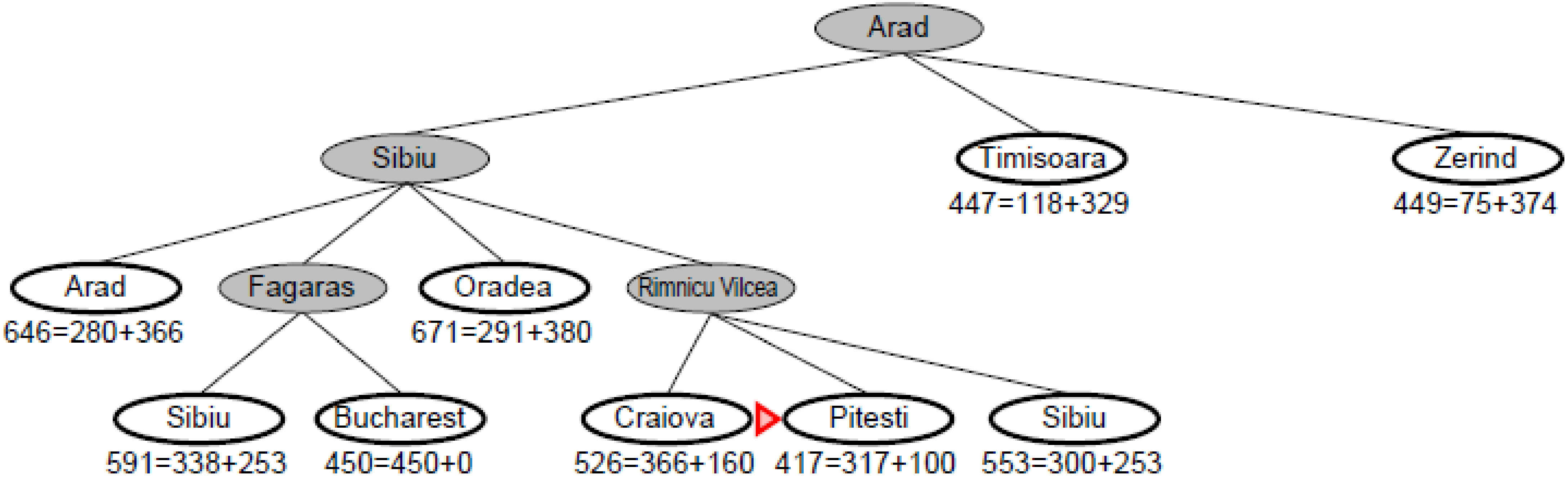
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193



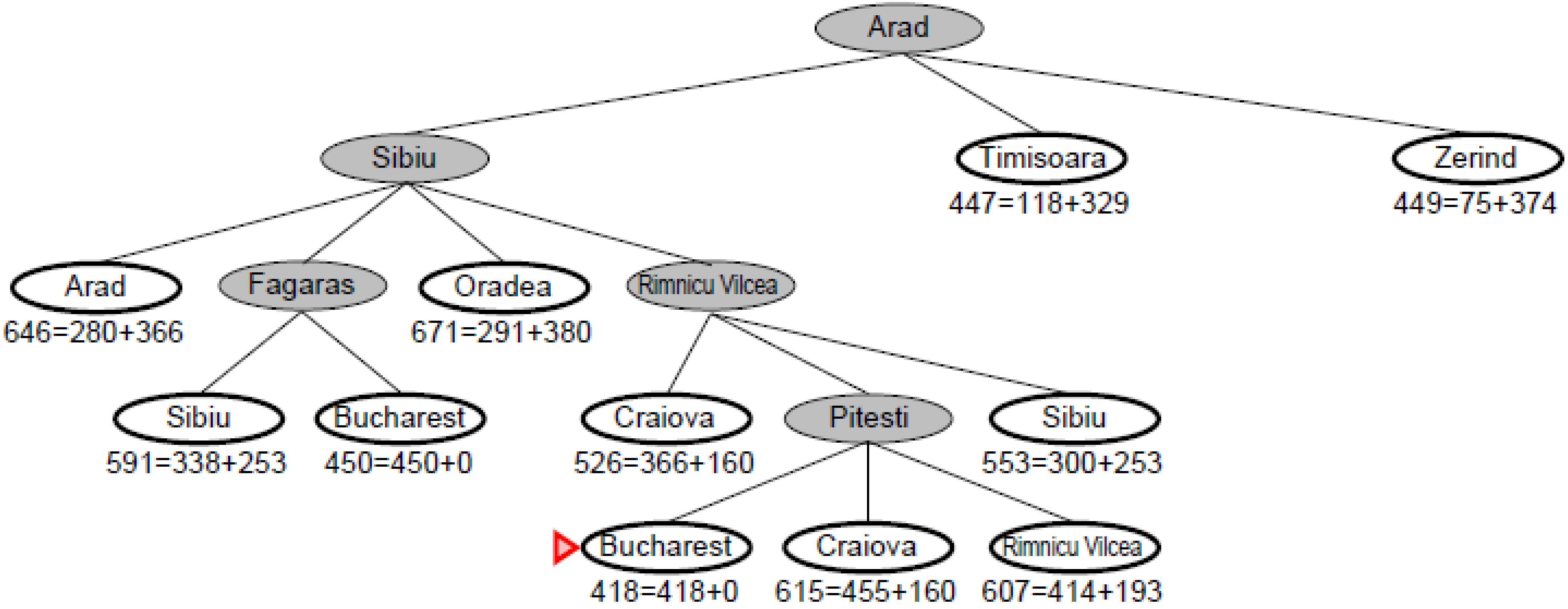
A*: Illustration 2



A*: Illustration 2



A*: Illustration 2



A*: Pseudo-code

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

Recap: Core Idea

- Idea: avoid expanding paths that are already expensive.
- Expand the node s that has the minimal $f(s) = h(s) + g(s)$
  - $g(s)$: cost from Start to s.
  - $h(s)$: estimated cost from s to Goal.
  - ➢ $f(s)$: estimated total cost of path from Start through s to Goal.

Optimality of A*

- Theorem: A* with h is optimal if h is admissible.
- Proof:
  - **Notation**: S – start, G – goal, s – a node on optimal path, s' – non-optimal goal, c^* – cost of optimal path.
  - **To show**: A* always pick s over s' $\Leftrightarrow f(s) < f(s')$.
  - **Known**: h is admissible $\Rightarrow h(s) < c^*(s, G)$.
  - **Deduce**:
    1) $f(s') = g(s') + h(s') = g(s') + 0 > c^*$ (s' is the goal node & c^* the smallest).
    2) $f(s) = g(s) + h(s) < g(s) + c^*(s, G) = c^*(S, s) + c^*(s, G) = c^*$.
  - **Conclude**: combine (1)&(2) $\Rightarrow f(s) < f(s')$ □

A*: Performance Metrics

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

- Complete? Yes.
- Optimal? Yes*, if h is admissible.
- Time? $O(b^d)$.
- Space? $O(b^d)$, keep every node in memory.

➢ If a solution exists, A* will find the best.

➢ Memory is the major problem for A*.

A*: Example

- Super Mario played by a path-finding algorithm A*. The bot won the Mario AI competitions in 2009 (https://youtu.be/DlkMs4ZHHr8).

Brief Summary

Summary: Heuristic Search

- Core idea: expand the path that seems most promising.
- Node s with lowest $f(s) \rightarrow$ the most promising.
- Usually $f(s) = h(s) + ?$.

- Greedy: Incomplete & Not always optimal
- A*: Complete & Optimal

Summary: Search Methods with $f(s)$

- The choice of f determines the search methods.
- Uniform-cost search: $f(s) = g(s)$.
- Greedy best-first search: $f(s) = h(s)$.
- A* search: $f(s) = g(s) + h(s)$.

- $g(s)$: the path cost from Start to node s.
- $h(s)$: the estimated cost from node s to Goal.

III. Further Studies on Heuristics

- Search Efficiency of Heuristics
- Generate Admissible Heuristics

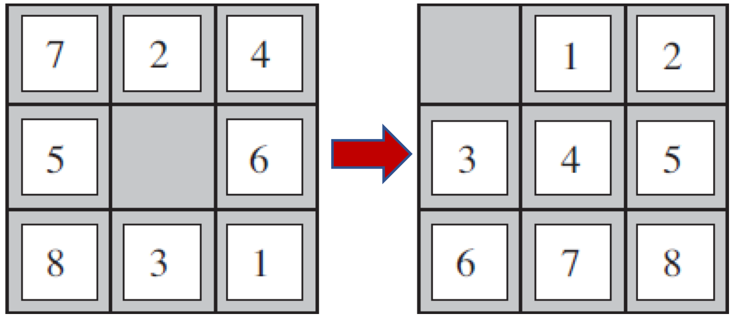
III.1 Search Efficiency of Heuristics

Recall: Heuristics for 8-puzzle Problem

- $h_{mis}(s)$ = #misplaced tiles $\in [0,8]$: Admissible.
- $h_{1stp}(s)$ = #(1-step move) to reach the goal configuration: Admissible.

➢ $h_{1stp}(s) \geq h_{mis}(s) \Rightarrow h_{1stp}(s)$ is ‘better’ than $h_{mis}(s)$.

What does ‘better’ mean?



Dominance

- For admissible h_1 and h_2, if $h_1(s) \geq h_2(s)$ for $\forall s$

  $\Rightarrow h_1$ dominates h_2 and is more efficient for search.
- Theorem: For any admissible heuristics h_1 and h_2, define

$$h(s) = \max\{h_1(s), h_2(s)\}$$

$h(s)$ is admissible and dominates both h_1 and h_2.

➢ 'Better' heuristic = dominance = better search efficiency.

Even Better Dominance

- Question: Which one to choose from a collection of admissible heuristics $h_1, \cdots, h_m$ & none dominates any other?
- Answer: $h(s) = \max\{h_1(s), \cdots, h_m(s)\}$ dominates all the others.

Effect of Heuristic Accuracy on Performance

- Quality indicator of an heuristic:
  - Effective branching factor b^*
    - $N =$ total number of nodes
    - $d =$ solution depth
    - Then, $N + 1 = 1 + \mathrm{b}^* + (\mathrm{b}^*)^2 + \ldots + (\mathrm{b}^*)^{\mathrm{d}}$
    - A well designed heuristic would have a b^* close to 1, allowing to solve a large problem at reasonable cost.

Quantify Search Efficiency

- Effective Branching Factor b^*: For a solution from A*, calculate b^* satisfying: $$N = b^* + (b^*)^2 + \cdots + (b^*)^d$$
  - N: #nodes of the solution,
  - d: depth of the solution tree.
- E.g., A* finds a solution at depth 5 using 52 nodes ⇒ $b^* = 1.92$.
- Good heuristics have b^* close to 1 → large problems solved at reasonable computational cost.

➢ b^* quantifies search efficiency of heuristics.

Empirical: Factor b^* for Search Efficiency

- Aim: Compare h_1 and h_2 regarding the search efficiency.
- Setting: Generate 1200 random problems with $d = \{2, \cdots, 24\}$ and solve them with IDS and A* with h_1 & h_2.
- Note: IDS – a baseline.

Empirical: Factor b^* for Search Efficiency

	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

Empirical: Factor b^* for Search Efficiency

	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						
6						
8						
10						
12						
14	–	[illegible]	[illegible]	–	[illegible]	[illegible]
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

- h_2 is 'better' than h_1 regarding search efficiency.
- This goodness is reflected by b^* being closer to 1.
- A* with h_2 performs much better than IDS.

III.2 Generate Admissible Heuristics

We Know about Heuristics ...

- We know:
  - How to judge their admissibility.
  - How to compare their goodness regarding searching efficiency.

- Question: How to produce such ‘good’ heuristics?

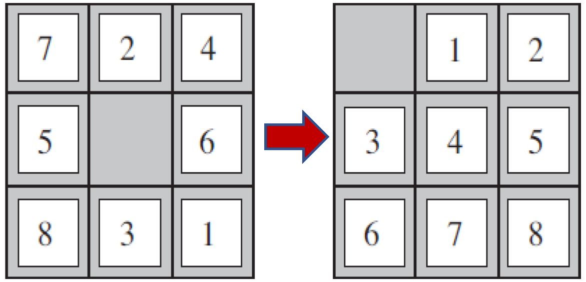
(1) Generate from Relaxed Problems

Where are h_{mis} & h_{1stp} from?

For 8-puzzle problem:

- Real Rule: A tile can only move to the adjacent empty square.
- Relaxed rules: h_{mis} and h_{1stp} are admissible
  - R1: A tile can move anywhere $\Rightarrow h_{mis}(s) = \#$(misplaced tiles).
  - R2: A tile can move one step in any direction regardless of an occupied neighbour $\Rightarrow h_{1stp}(s) = \#$(1-step move) to reach goal.

➢Optimal solutions to problems with R1, R2 are easier to find.



Relaxed Problem

- Relaxed problem: a problem with relaxed rules on the action.
- E.g. 8-puzzle problems with R1 and R2.

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

➢ No wonder h_{mis} and h_{1stp} are admissible.

(2) Generate from Sub-problems

Subproblem

- Subproblem
  - Task: get tiles 1, 2, 3 and 4 into their correct positions.
  - Relaxation: move them disregarding the others.
- Theory: cost*(subproblem)<cost*(original).
  - cost*(subproblem): the cost of the optimal solution of this subproblem.

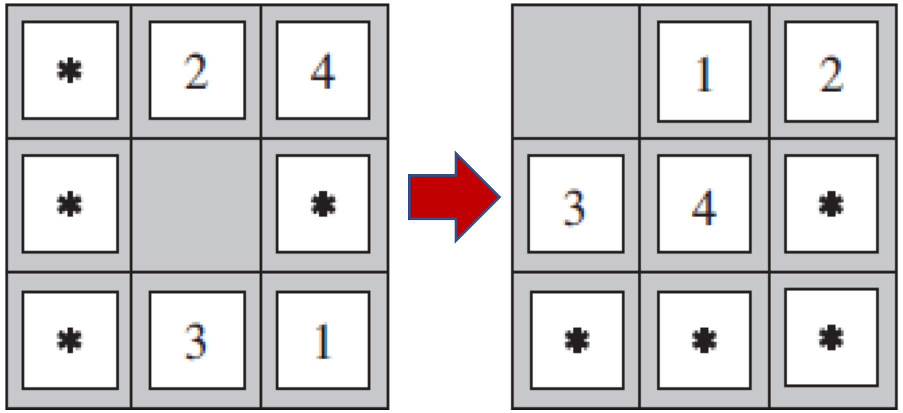


Fig.1. A subproblem of 8-puzzle.

Subproblem and Admissible Heuristics

- Admissible $h^*_{sub}(s)$: estimate the cost from s to the subproblem goal.
  - E.g. $h_{sub}^{(1,2,3,4)}$ is the cost to solve the 1-2-3-4 subproblem.

- $h_{sub}(s)$ dominates $h_{1stp}(s)$
  - $h_{sub}(s) = max\{h_{sub}^{(1,2,3,4)}(s), h_{sub}^{(2,3,4,5)}(s), \cdots\}$.

Disjoint Subproblems

- Question: Will the addition of heuristics from subproblem (1-2-3-4) and (5-6-7-8) give an admissible heuristic, considering the two subproblems are not overlapped?
- Answer: No, since they always share some moves.

- Question: What if not count those shared moves?
- Answer: $h_{sub}^{(1,2,3,4)}(s) + h_{sub}^{(5,6,7,8)}(s) \leq c^*(s) \Rightarrow$ admissible.
  - Disjoint pattern database

(3) Generate from Experiences

'Experience' Formulation

For 8-puzzle problem:

- Solve many 8-puzzles to obtain many examples.
- Each example consists of a state from the solution path and the actual cost of the solution from that point.

➢ These examples are our 'experience' for this problem.

- Question: How to learn $h(s)$ from these experience?

Learn Heuristics from Experience

- Question: What are the good experience features?
- Answer: Relevant to predicting the states' cost to Goal, e.g.
  - $x_1(s)$: #(displaced tiles).
  - $x_2(s)$: #(pairs of adjacent tiles) that are not adjacent in Goal state.
- Question: How to learn h from those relevant experience features?
- Answer: (e.g.) Construct model as

$$h(s) = w_1 x_1(s) + w_2 x_2(s),$$

where w_1, w_2 are model parameters to learn from training data by a learning method such as neural networks and decision trees.

Brief Summary

Summary: Search Methods

- **Uninformed Search**: Use no domain knowledge.
  - Other names: basic search, blind search
  - BFS, UCS, DFS, DLS and IDS

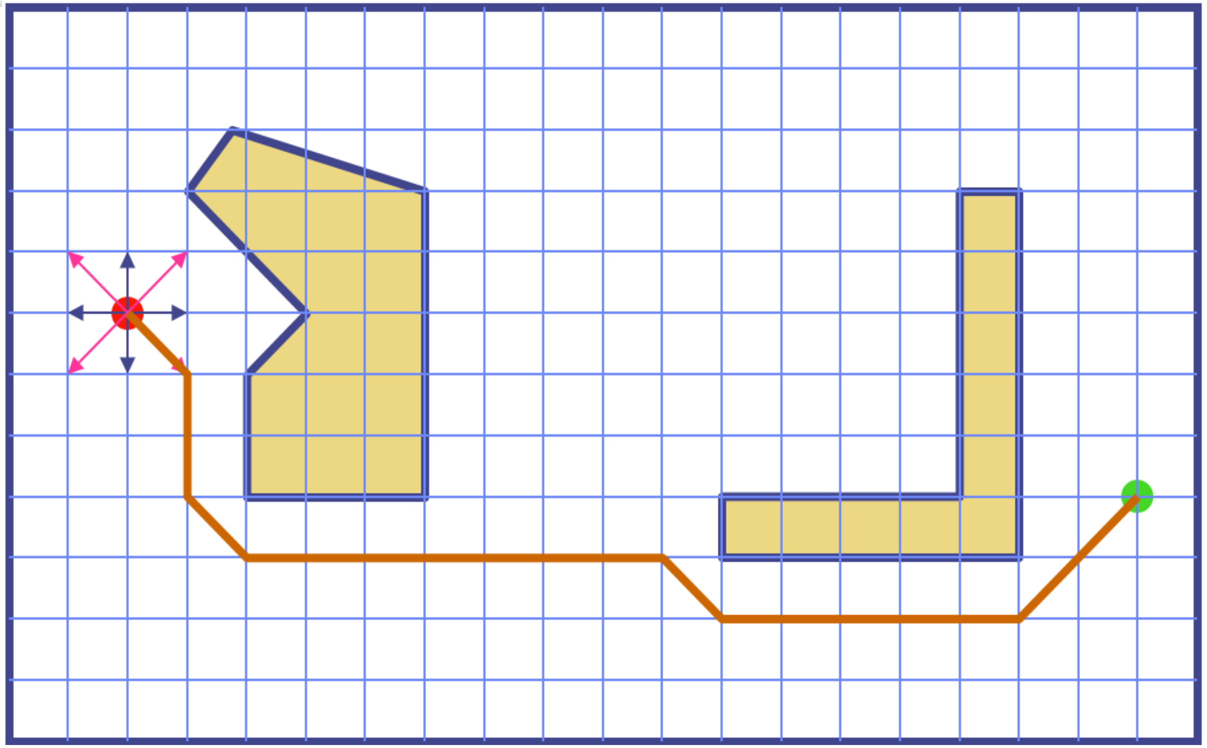
- **Informed Search**: Use heuristics to encode domain knowledge.
  - Other name: heuristic search
  - Greedy search and A* search.

Summary: Heuristics

- Heuristic $h(s)$: estimate the cost from s to Goal.
  - (1) $h(s) = 0$ if s is Goal. (2) nonnegative. (3) problem-specific.
- Admissibility: $h(s)$ never overestimates cheapest cost from s to Goal.
- Effective branching factor b^*: quantify the search efficiency of $h(\cdot)$.
- Generate admissible heuristics:
  - (1) from relaxed problems.
  - (2) from sub-problem.
  - (3) from experience.

Example: Robot Navigation

- Move along the line or diagonal.
- $h_{SLD}(s)$: straight-line distance from n to the goal.
- cost of a horizontal/vertical move = 1
- cost of one diagonal move = $\sqrt{2}$.



Question: find the optimal path from Red to Green by (1) greedy search and (2) A* search.

Reading Materials for This Lecture

- AI textbook (3rd edition)
  - Chapter II.3: Solving Problems by Searching (pages 92-108)
  - AI-book codes: https://github.com/aimacode
- Algorithms textbook
  - 24: Single-Source Shortest Paths (pages 644-683)
- Articles
  - [1] Seet, B. C., Liu, G., Lee, B. S., Foh, C. H., Wong, K. J., & Lee, K. K. (2004, May). A-STAR: A mobile ad hoc routing strategy for metropolis vehicular communications. In *International Conference on Research in Networking* (pp. 989-999). Springer, Berlin, Heidelberg.
  - [2] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). *Path planning with modified a star algorithm for a mobile robot*. Procedia Engineering, 96, 59-69.
- Demos:
  - http://www.cnblogs.com/0zcl/p/6242790.html
  - http://ashblue.github.io/javascript-pathfinding/
  - https://github.com/bgrins/javascript-astar
  - https://www.mathworks.com/matlabcentral/fileexchange/66461-astar-demo