Artificial Intelligence

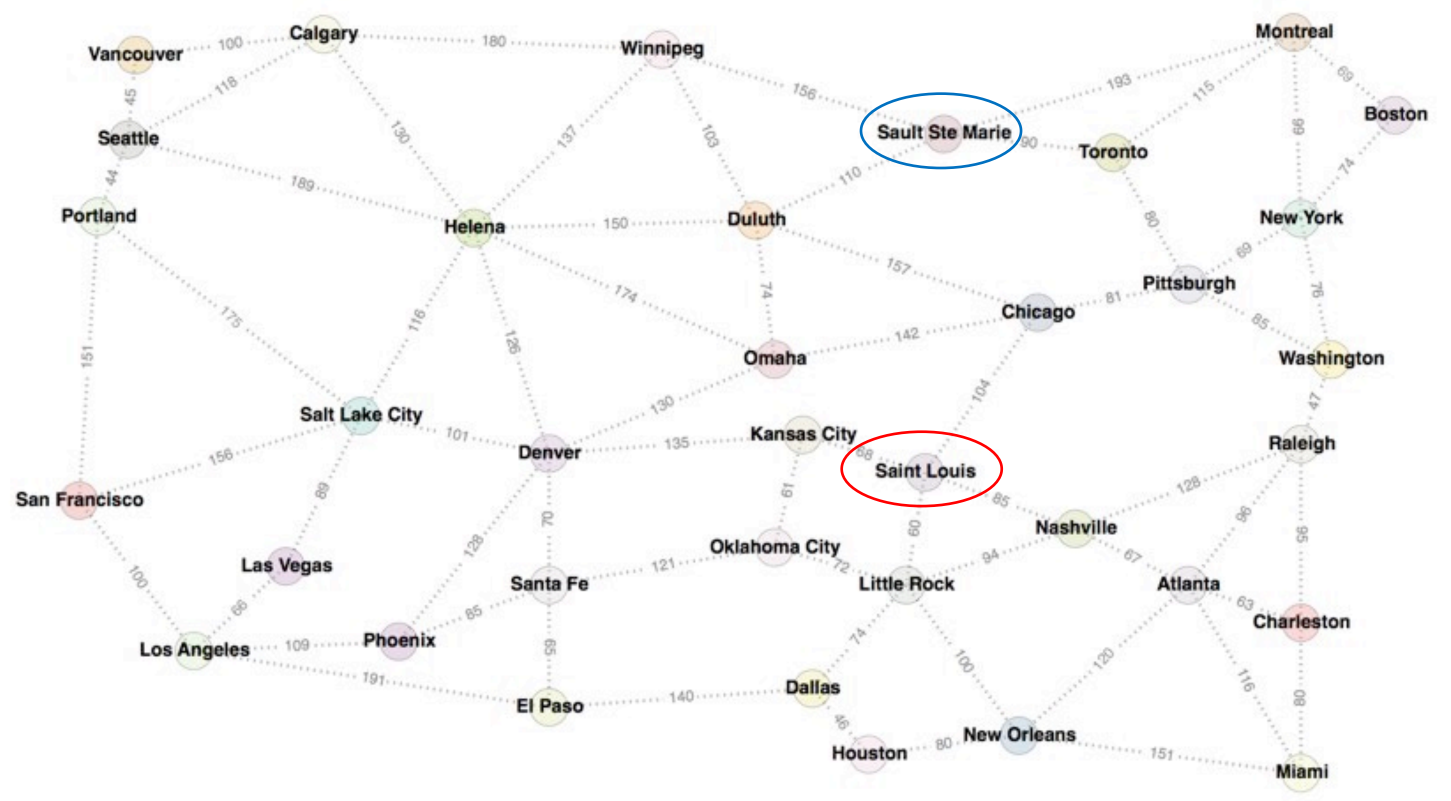
Lecture 4: Informed Search

Credit: Ansaf Salleb-Aouissi, and "Artificial Intelligence: A Modern Approach", Stuart Russell and Peter Norvig, and "The Elements of Statistical Learning", Trevor Hastie, Robert Tibshirani, and Jerome Friedman, and "Machine Learning", Tom Mitchell.

Informed search

- **Use domain knowledge!**
  - Are we getting close to the goal?
  - Use a heuristic function that estimates how close a state is to the goal
  - A heuristic does NOT have to be perfect!
  - Example of strategies:
    1. Greedy best-first search
    2. A* search
    3. IDA*

Informed search



The distance is the straight-line distance. The goal is to get to Sault Ste Marie, so all the distances are from each city to Sault Ste Marie.

Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Heuristic!

Greedy search

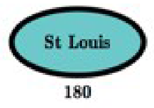
- Evaluation function $h(n)$ (heuristic)
- $h(n)$ estimates the cost from n to the goal
- Example: $h_{\text{SLD}}(n)$ = straight-line distance from n to Sault Ste Marie
- Greedy search expands the node that **appears** to be closest to goal

Greedy search: Pseudo-code

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost $f(n) = h(n)$ */

    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
```

Greedy search example

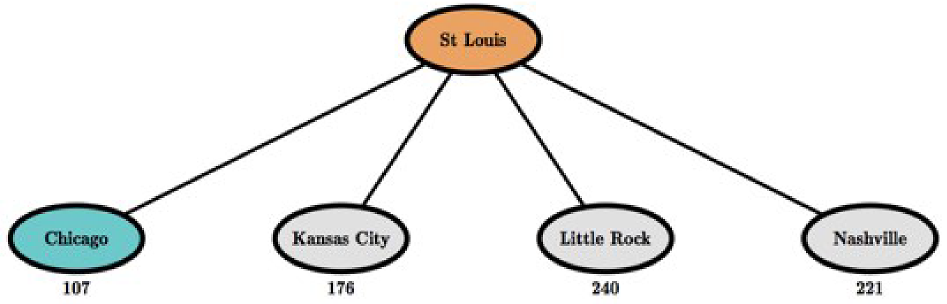
The initial state:



Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Greedy search example

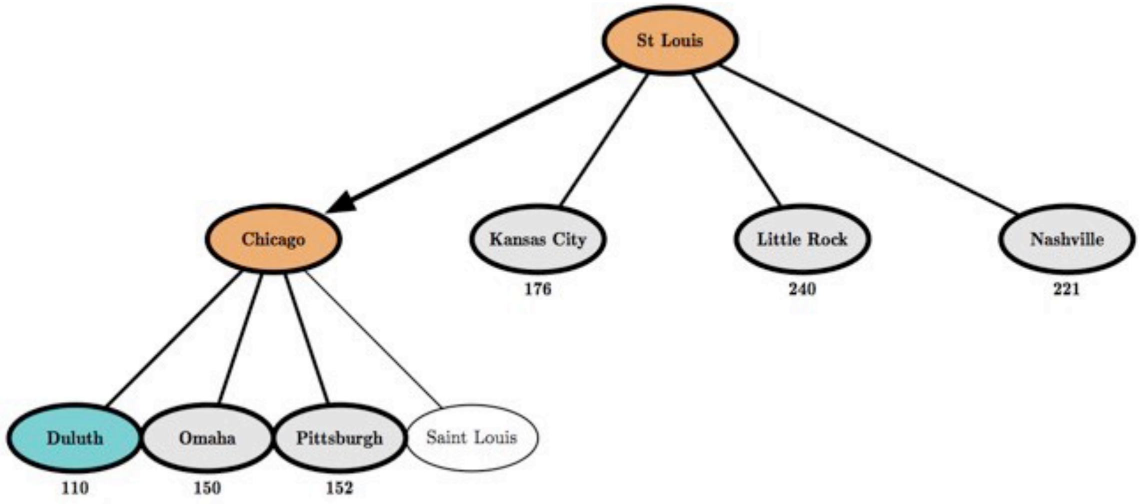
After expanding St Louis:



Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Greedy search example

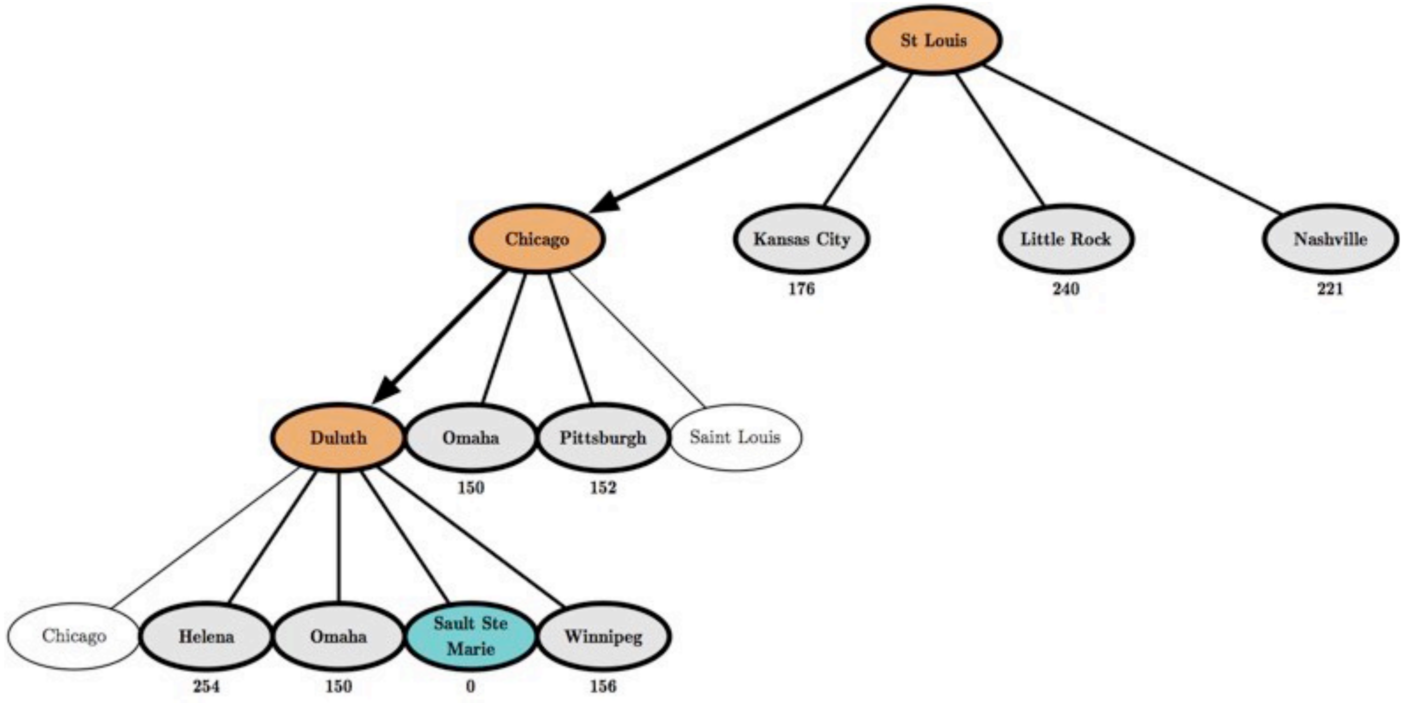
After expanding Chicago:



Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Greedy search example

After expanding Duluth:

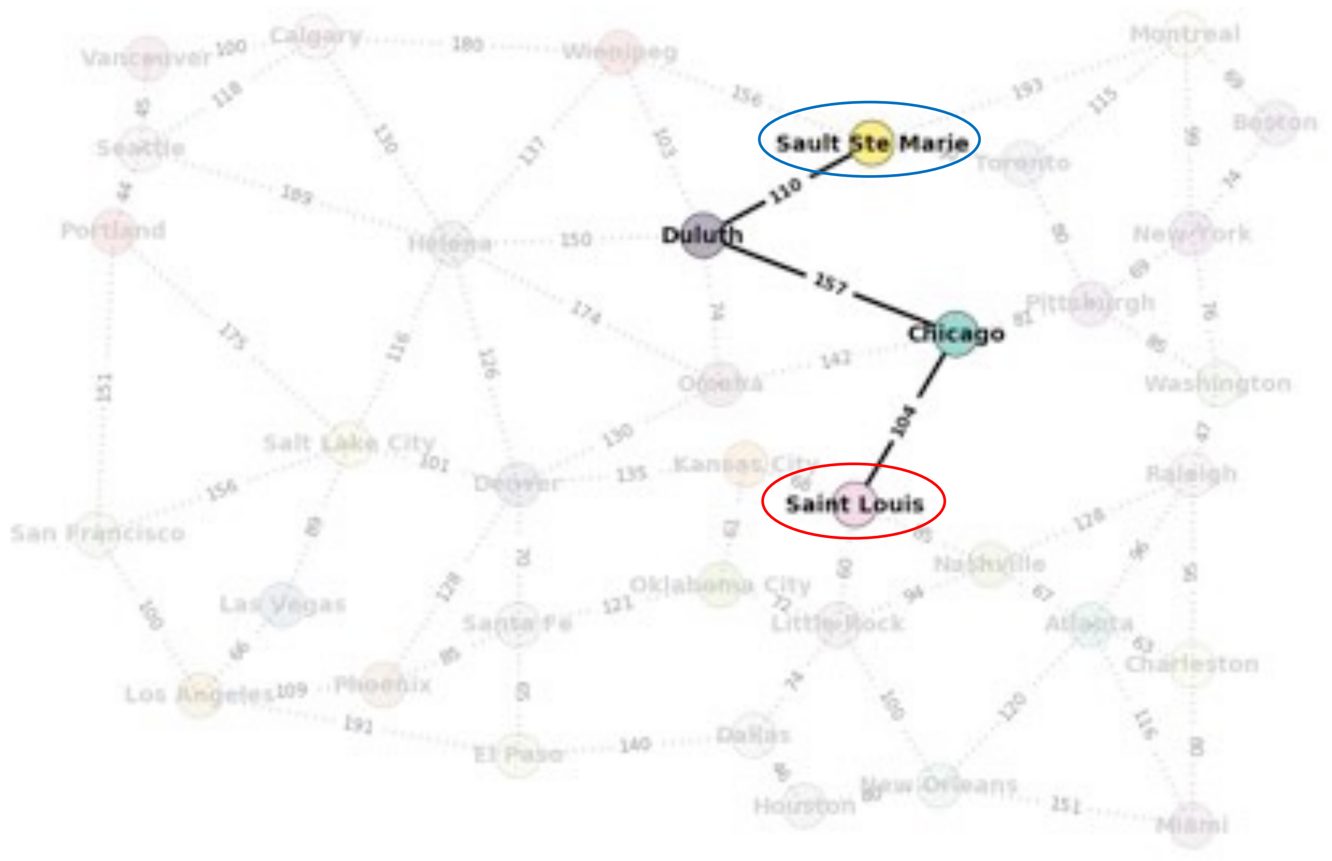


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Examples using the map (Greedy search)

Start: Saint Louis

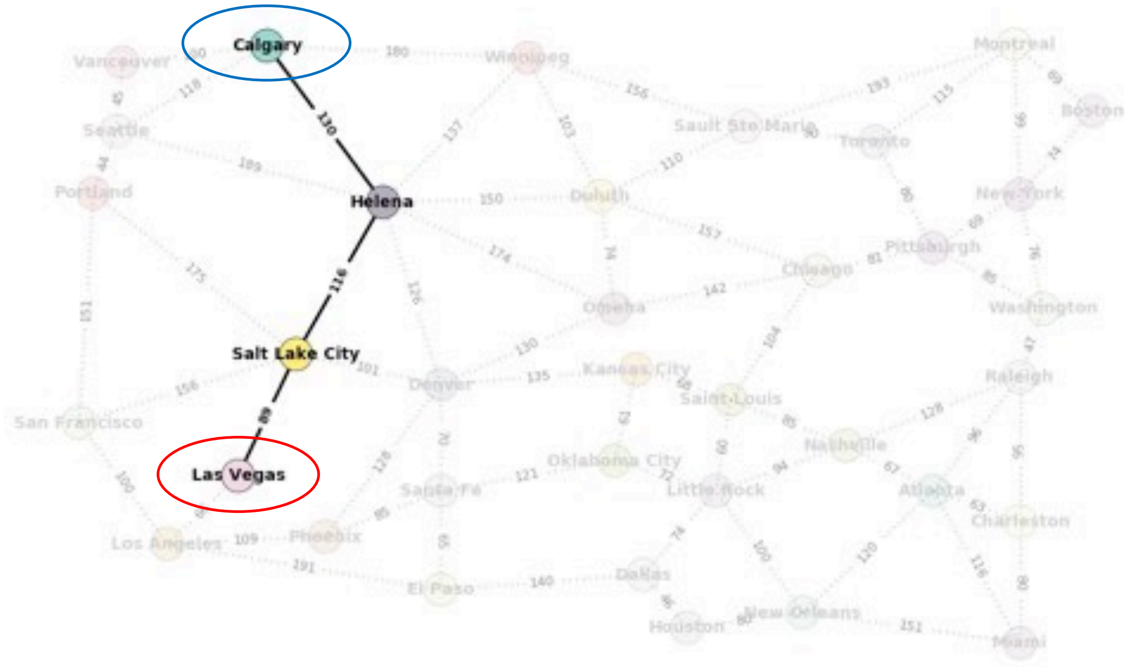
Goal: Sault Ste Marie



Examples using the map (Greedy search)

Start: Las Vegas

Goal: Calgary



A^* search

- Minimize the total estimated solution cost
- Combines:
  - $g(n)$: cost to reach node n
  - $h(n)$: cost to get from n to the goal
  - $f(n) = g(n)+h(n)$

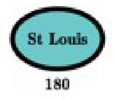
$f(n)$ is the estimated cost of the cheapest solution through n

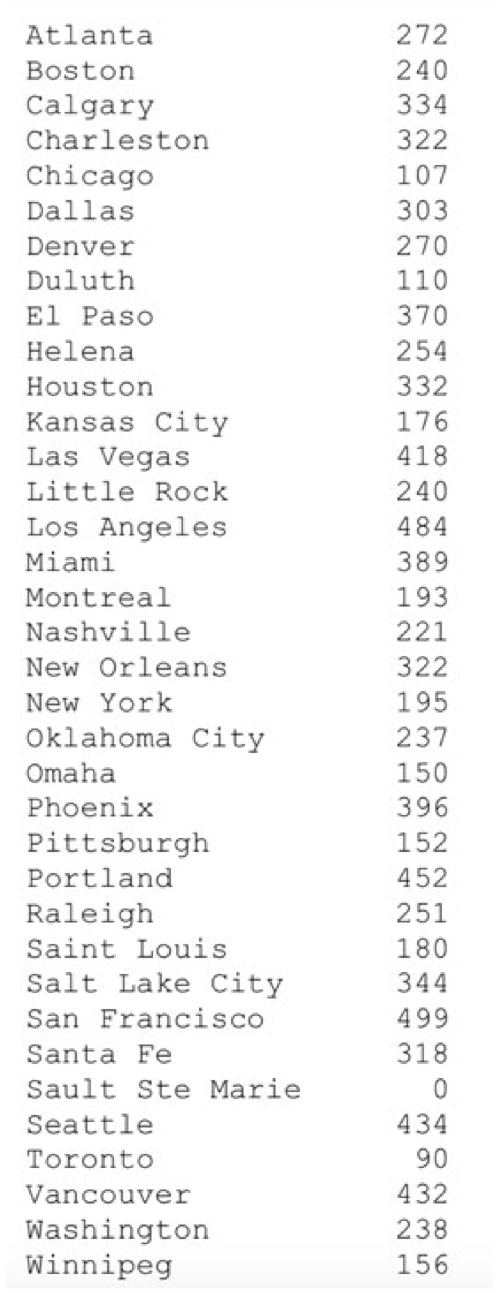
A* search: Pseudo-code

```
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 $\cup$ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

A* search example

The initial state:

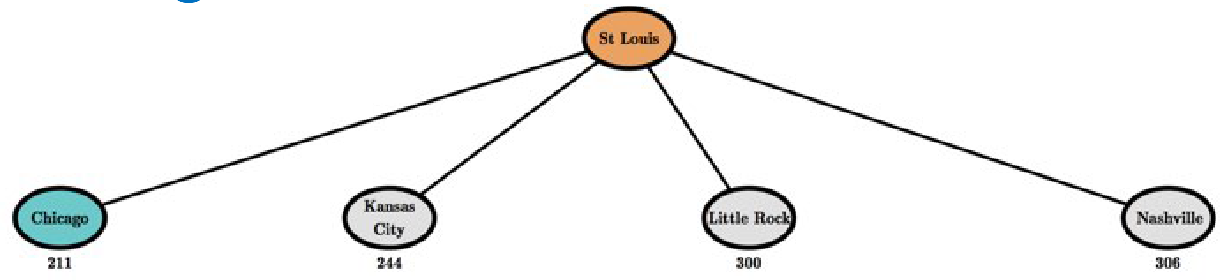




Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

After expanding St Louis:

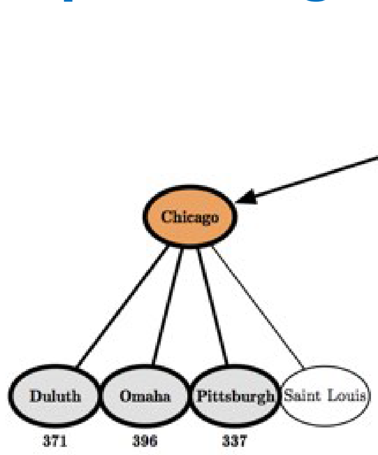


$g(n)$=104
$h(n)$=107
$f(n)$=211

Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

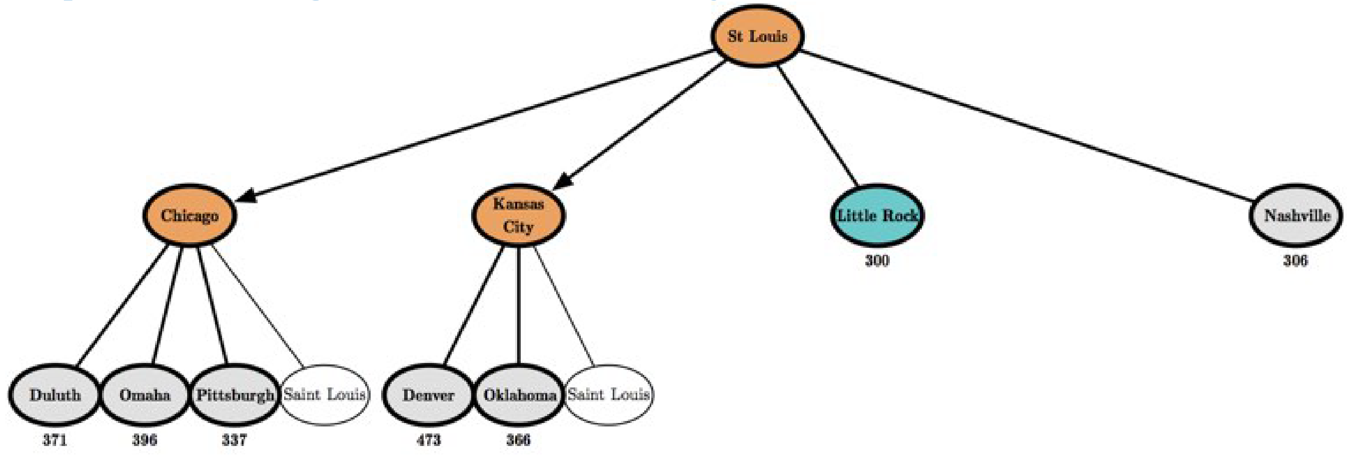
After expanding Chicago:

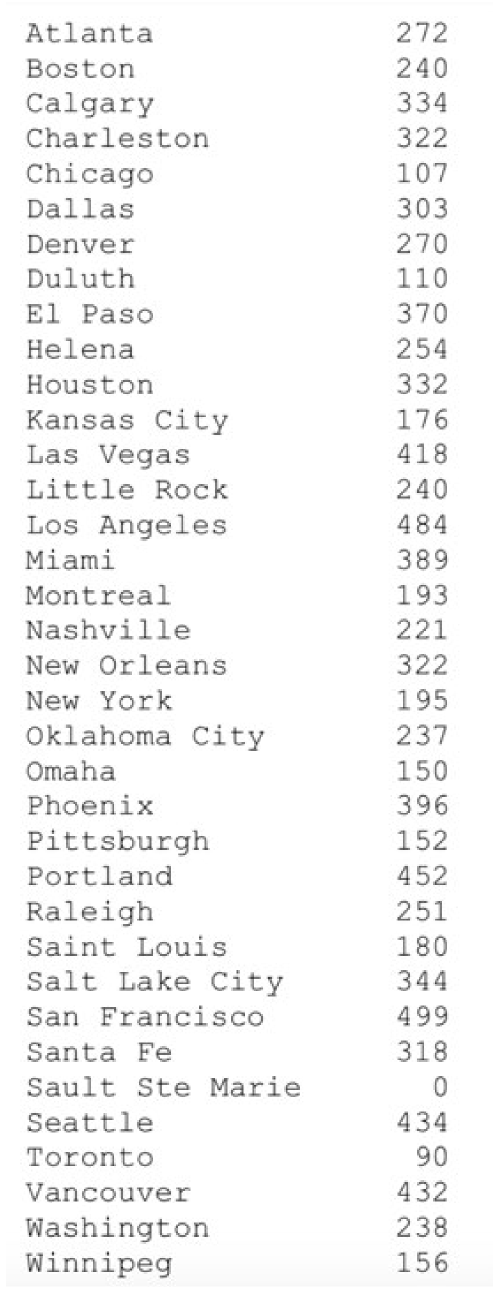


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

After expanding Kansas City:

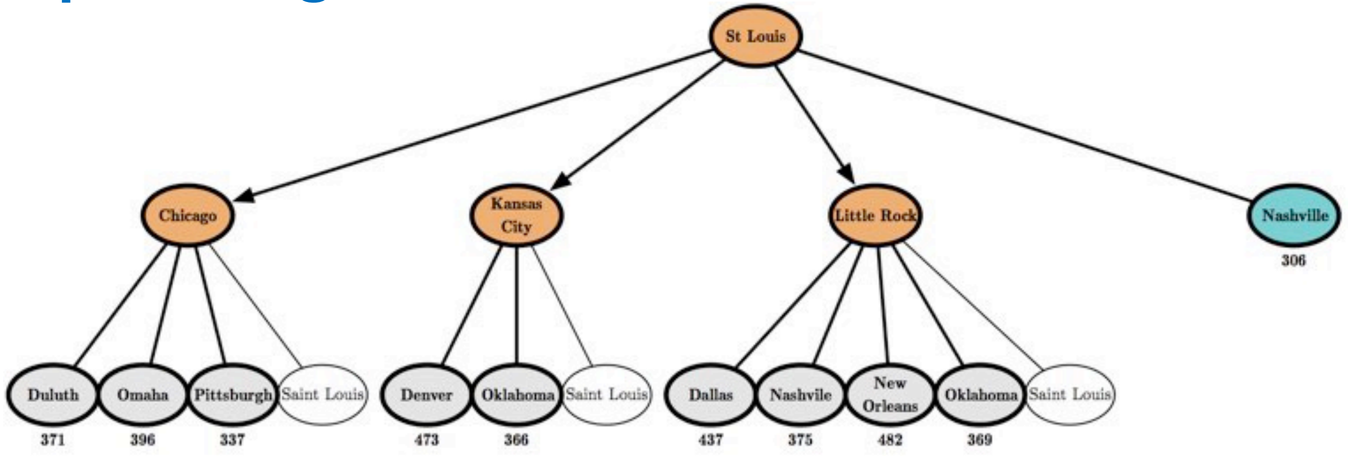


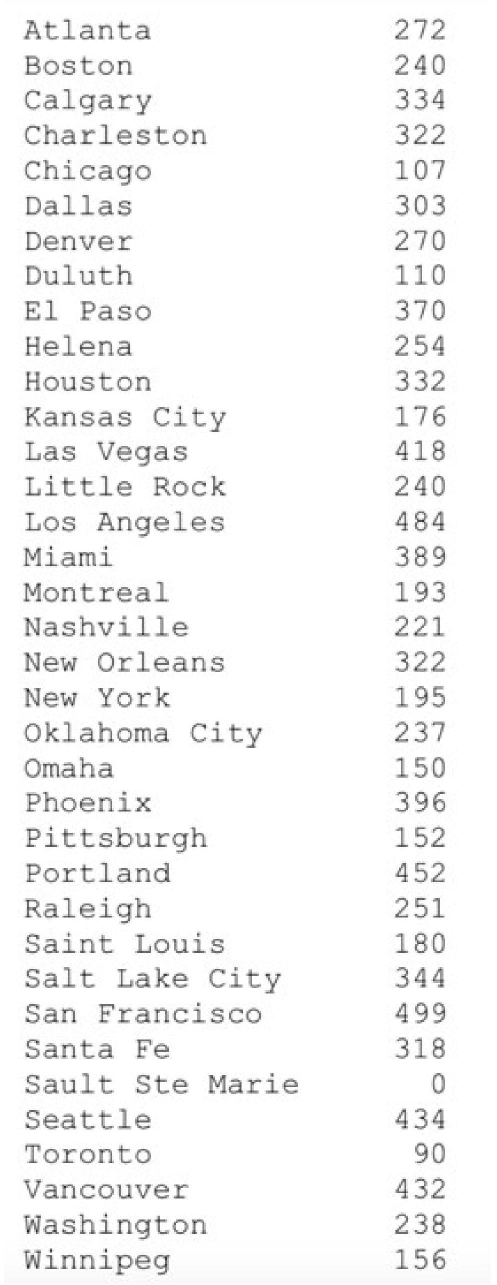


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

After expanding Little Rock:

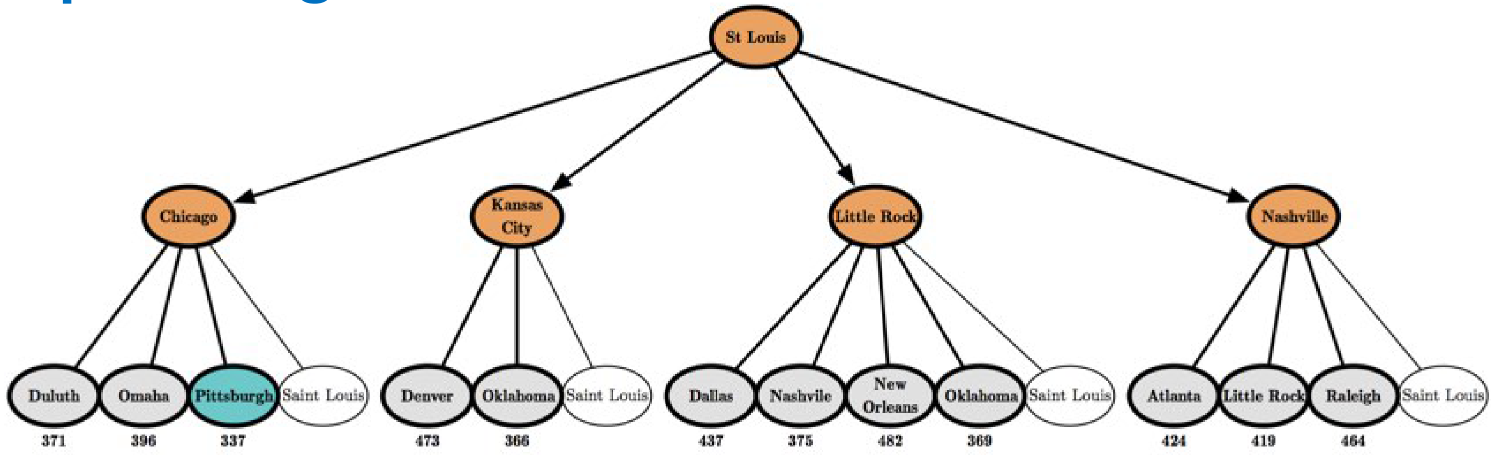




Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

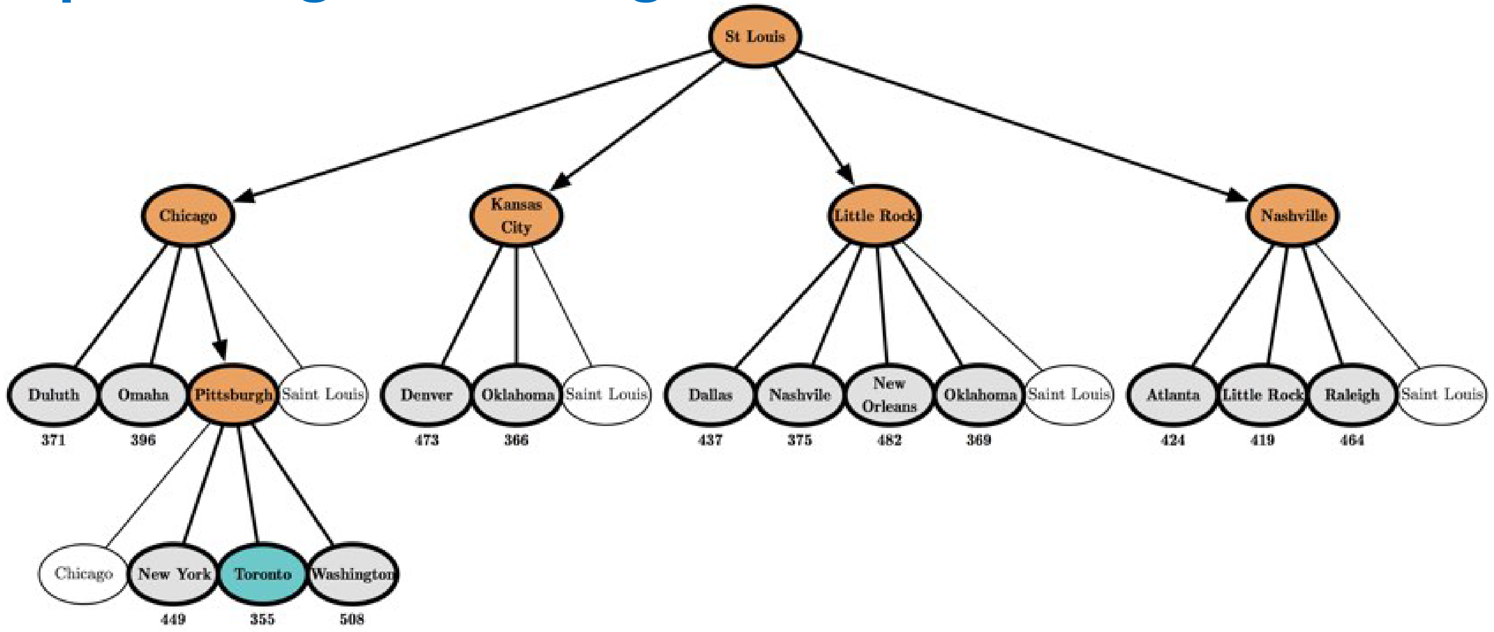
After expanding Nashville:

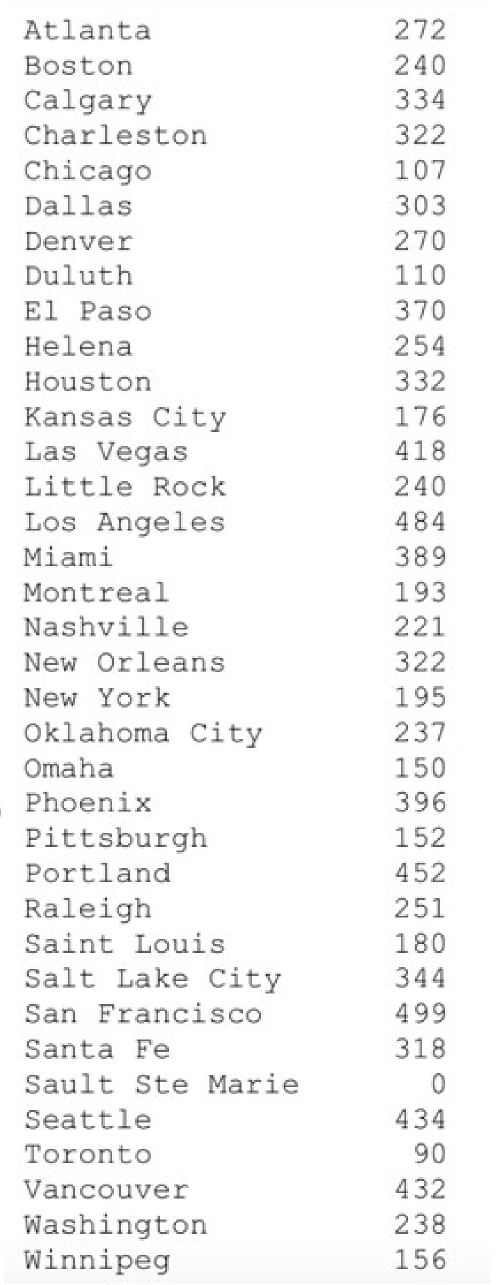


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

After expanding Pittsburgh:

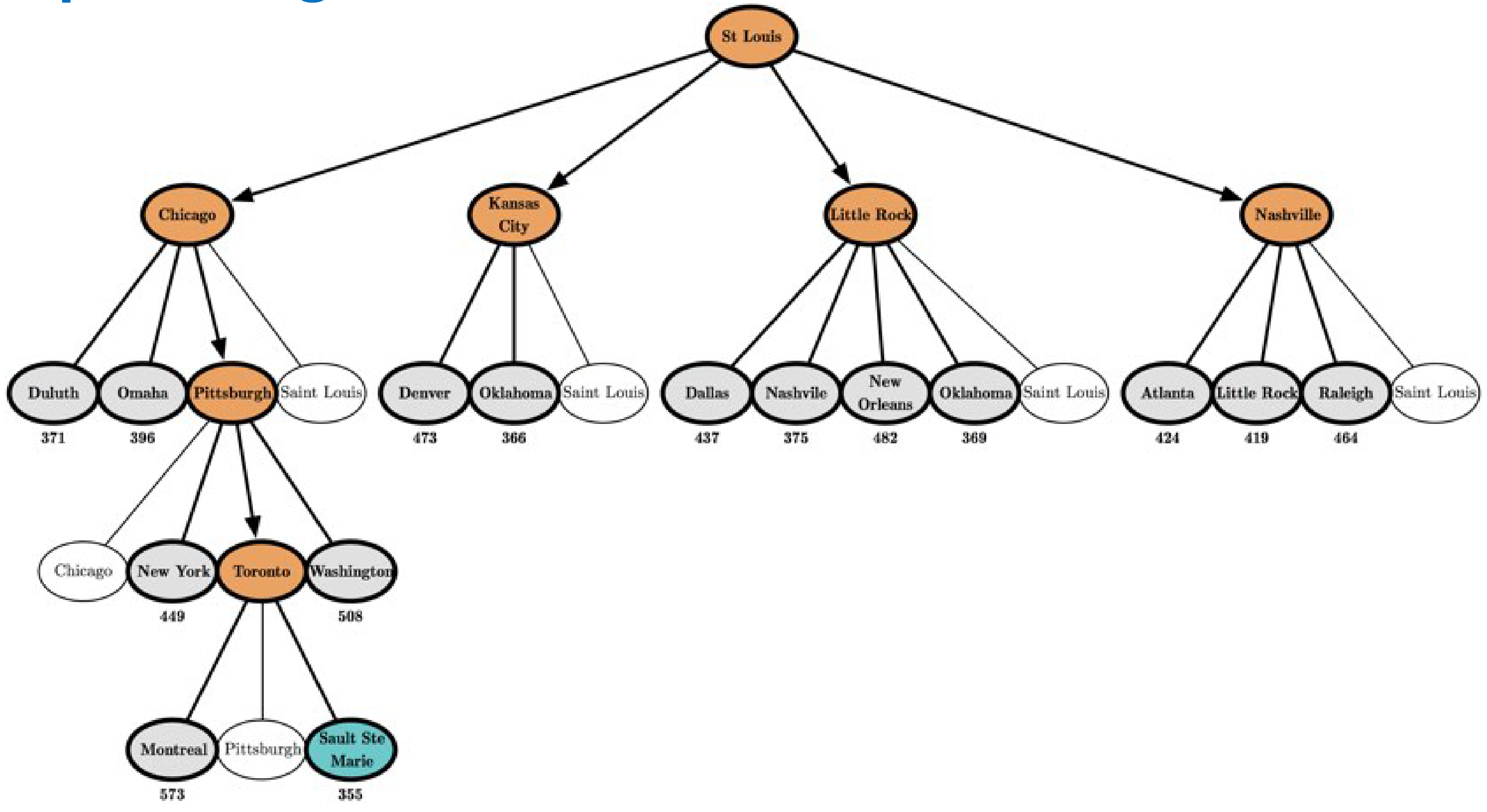




Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

A* search example

After expanding Toronto:

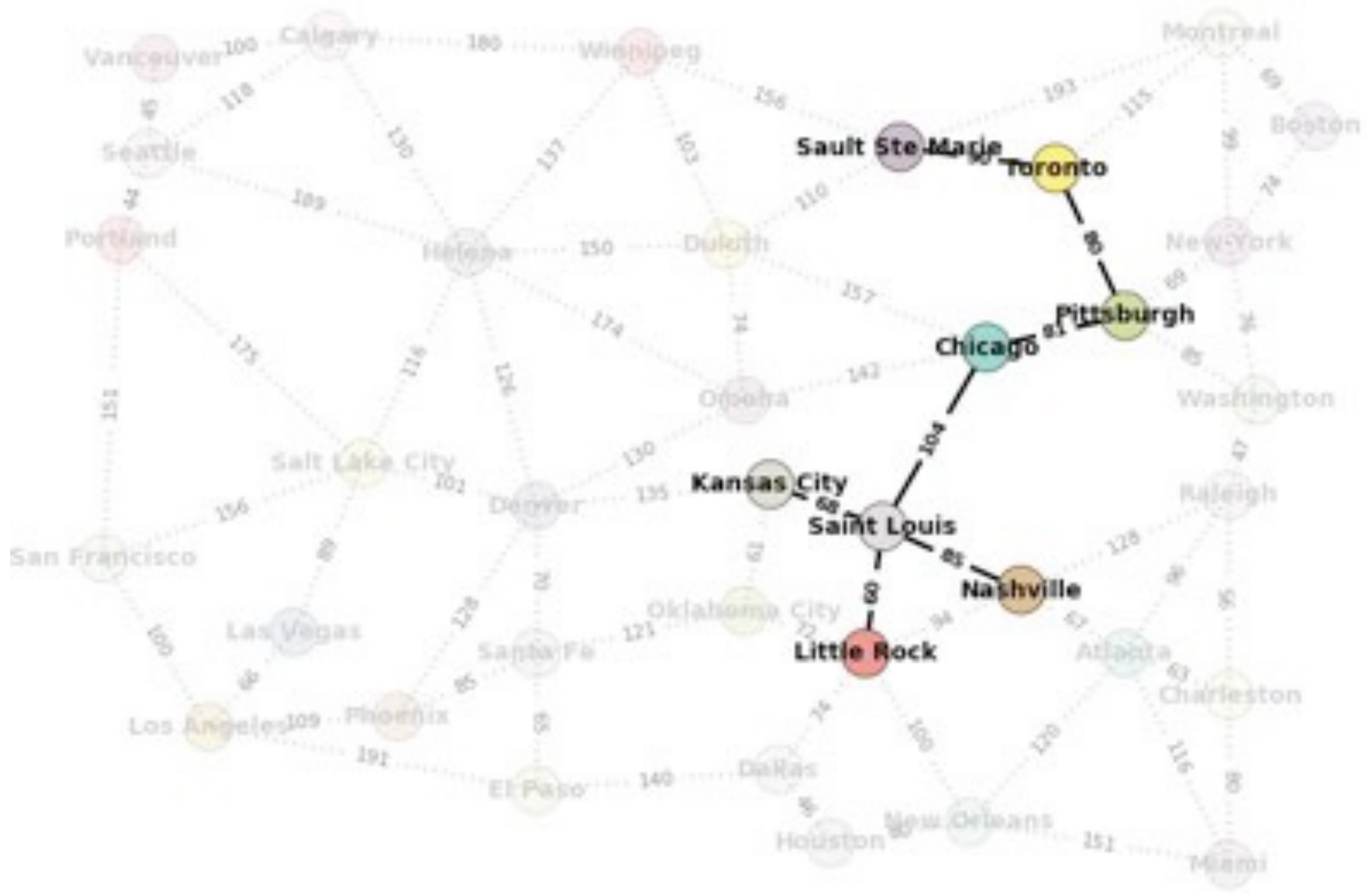


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Examples using the map (A* search)

Start: Saint Louis

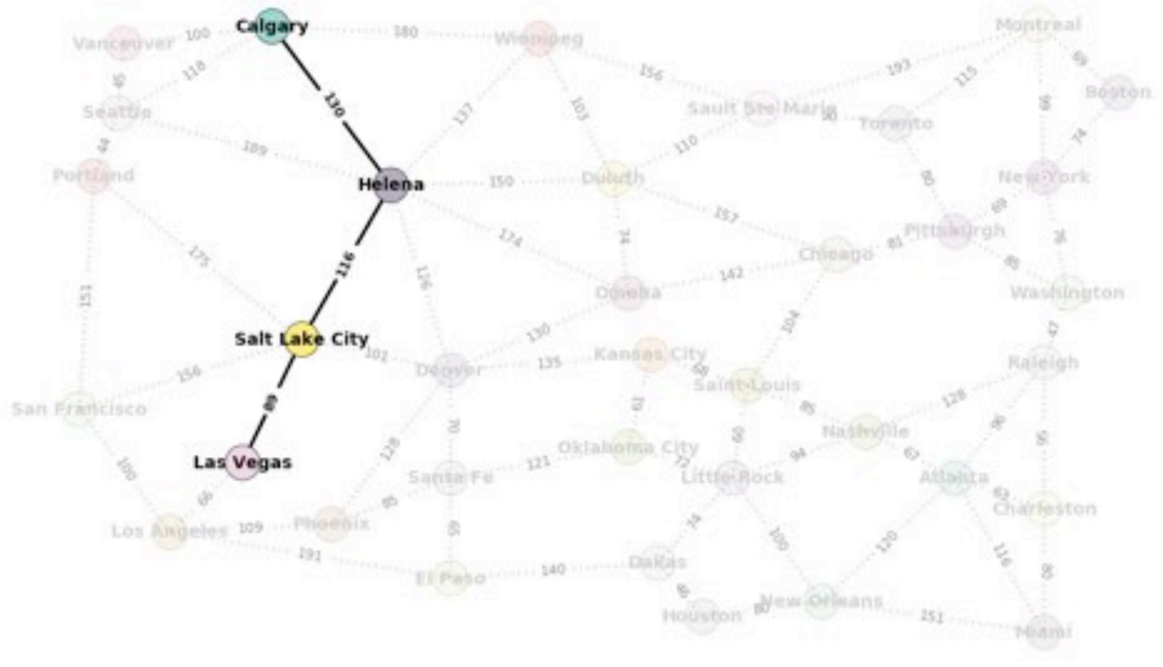
Goal: Sault Ste Marie



Examples using the map (A* search)

Start: Las Vegas

Goal: Calgary



Admissible heuristics

A good heuristic can be powerful.

Only if it is of a “good quality”

A good heuristic must be admissible.

Admissible heuristics

- An **admissible** heuristic never overestimates the cost to reach the goal, that is it is **optimistic**
- A heuristic *h* is admissible if

$$\forall node\ n,\ \ h(n) \leq h^*(n)$$

  where *h** is true cost to reach the goal from *n*.
- h_{SLD} (used as a heuristic in the map example) is admissible because it is by definition the shortest distance (straight line) between two points.

A* Optimality

If $h(n)$ is admissible, A* using tree search is optimal.

Rationale:

- Suppose G_o is the optimal goal.
  Suppose G_s is some suboptimal goal.
  Suppose n is on the shortest path to Go.
- $f(G_s) = g(G_s)$ since $h(G_s) = 0$
  $f(G_o) = g(G_o)$ since $h(G_o) = 0$
  $g(G_s) > g(G_o)$ since G_s is suboptimal
  Then **$f(G_s) > f(G_o)$ . . . (1)**
- $h(n) \leq h^*(n)$ since h is admissible
  $g(n)+h(n) \leq g(n)+h^*(n) = g(G_o) = f(G_o)$
  Then, **$f(n) \leq f(G_o)$ . . . (2)**

From (1) and (2) $f(G_s) > f(n)$, so A* will never select G_s during the search and hence A* is optimal.

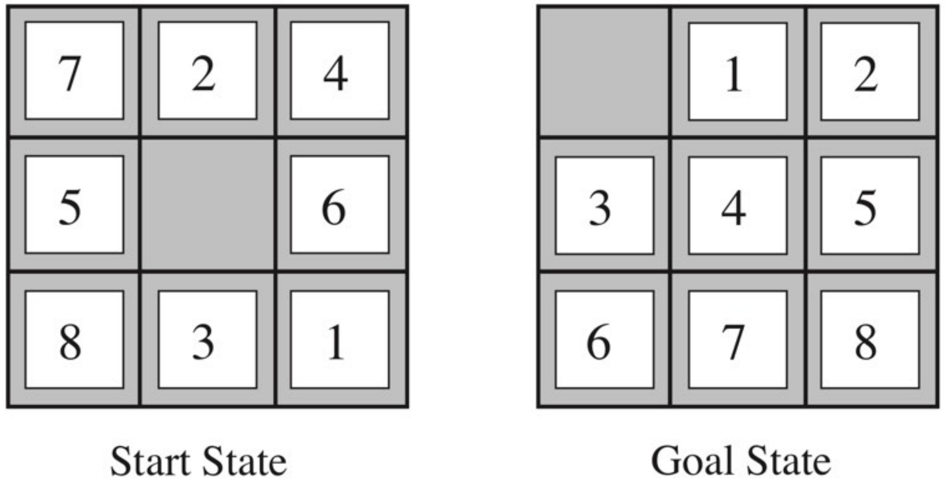
We always will go toward *Go* rather than to go *Gs*.

A* : PF Metrics

- **Complete:** Yes.
- **Time:** exponential
- **Space:** keeps every node in memory, the biggest problem
- **Optimal:** Yes!

Heuristics

- The solution is 26 steps long.
- $h_1(n)$ = number of misplaced tiles
- $h_1(n) = 8$
- $h_2(n)$ =total Manhattan distance (sum of the horizontal and vertical distances).
- Tiles 1 to 8 in the start state gives: $h_2 = 3+1+2+2+2+3+3+2 = 18$ which does not overestimate the true solution.



Further Studies on Heuristics

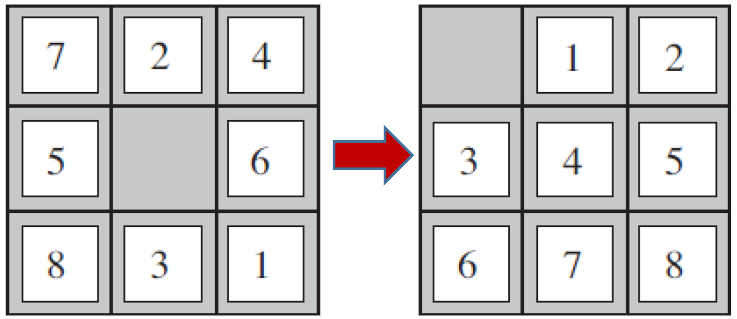
Ⅰ. Search Efficiency of Heuristics

Recall: Heuristics for 8-puzzle

- $h_{mis}(s)$ = #misplaced titles $\in [0,8]$: **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.

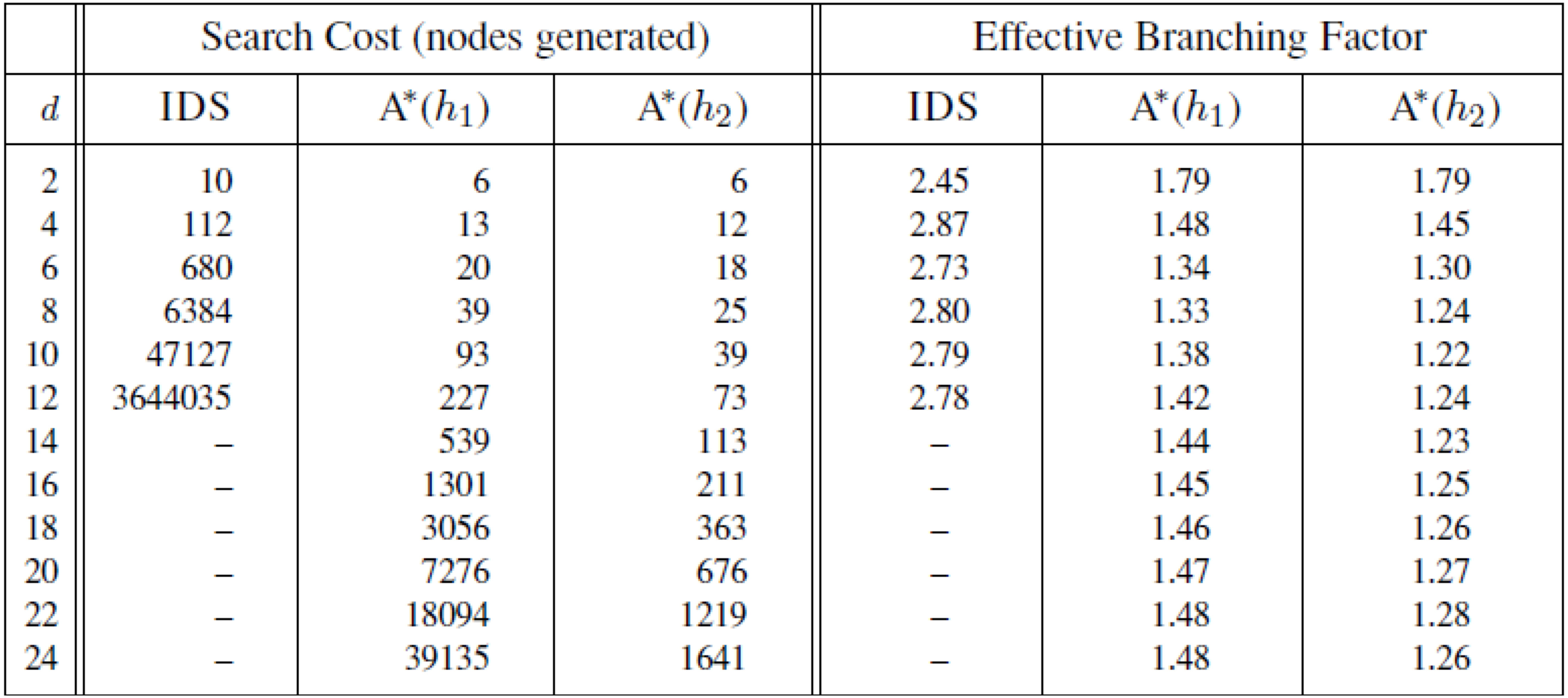
Quantify Search Efficiency

- **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 $\Rightarrow b^* = 1.92$.
- Good heuristics have b^* close to 1 $\Rightarrow$ large problems solved at reasonable computational cost.
- **b^* quantifies search efficiency of heuristics.**

Empirical: Factor b^*

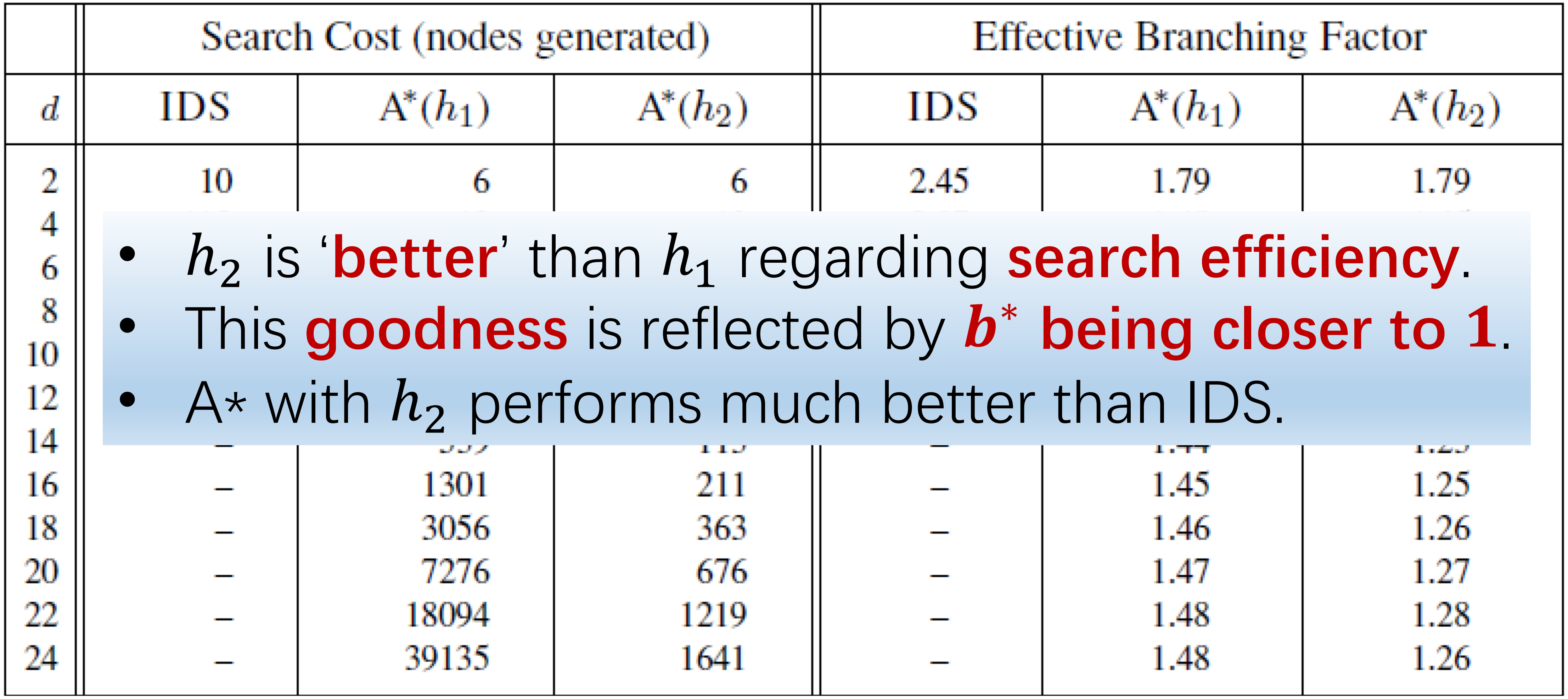
- **Aim**: Compare h_1 and h_2 regarding the search efficiency.
- **Setting**: Generate 1200 random problems with $d = \{2, \cdots, 24\}$ and solve them with IDS and A∗ with h_1 & h_2.
- **Note**: IDS – a baseline.

Empirical: Factor b^*



	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^*



	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	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
8	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
10	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
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.

II. Generate Admissible Heuristics

We Know about Heuristics ...

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

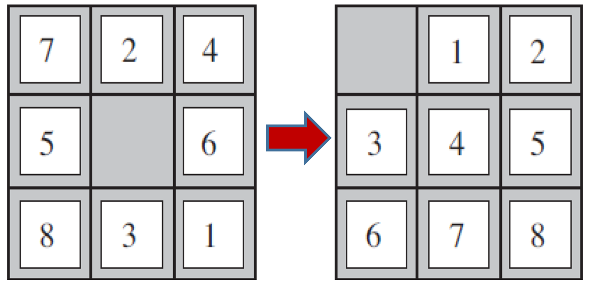
- **Question**: How to produce such ‘good’ heuristics?

(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 titles).
  - 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.

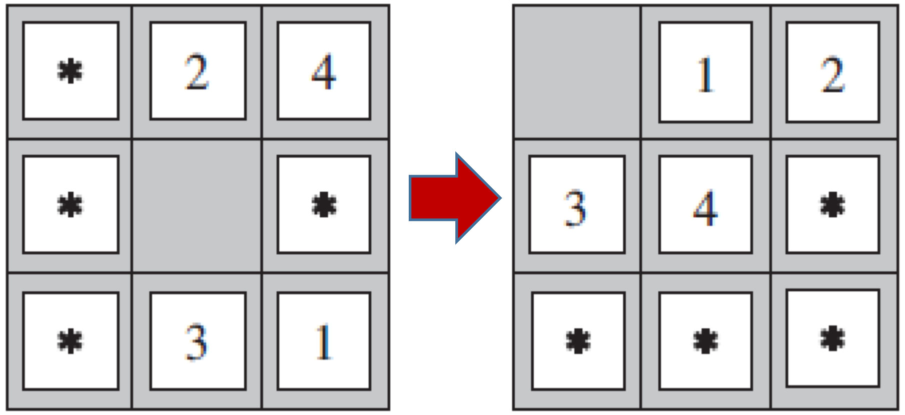


Fig.1. A subproblem of 8-puzzle.

Subproblem and Admissible Heuristics

- **Admissible $\boldsymbol{h^*_{sub}(s)}$**: 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.

- **Theorem**: $h_{sub}(s)$ 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.

To be continued