Artificial Intelligence

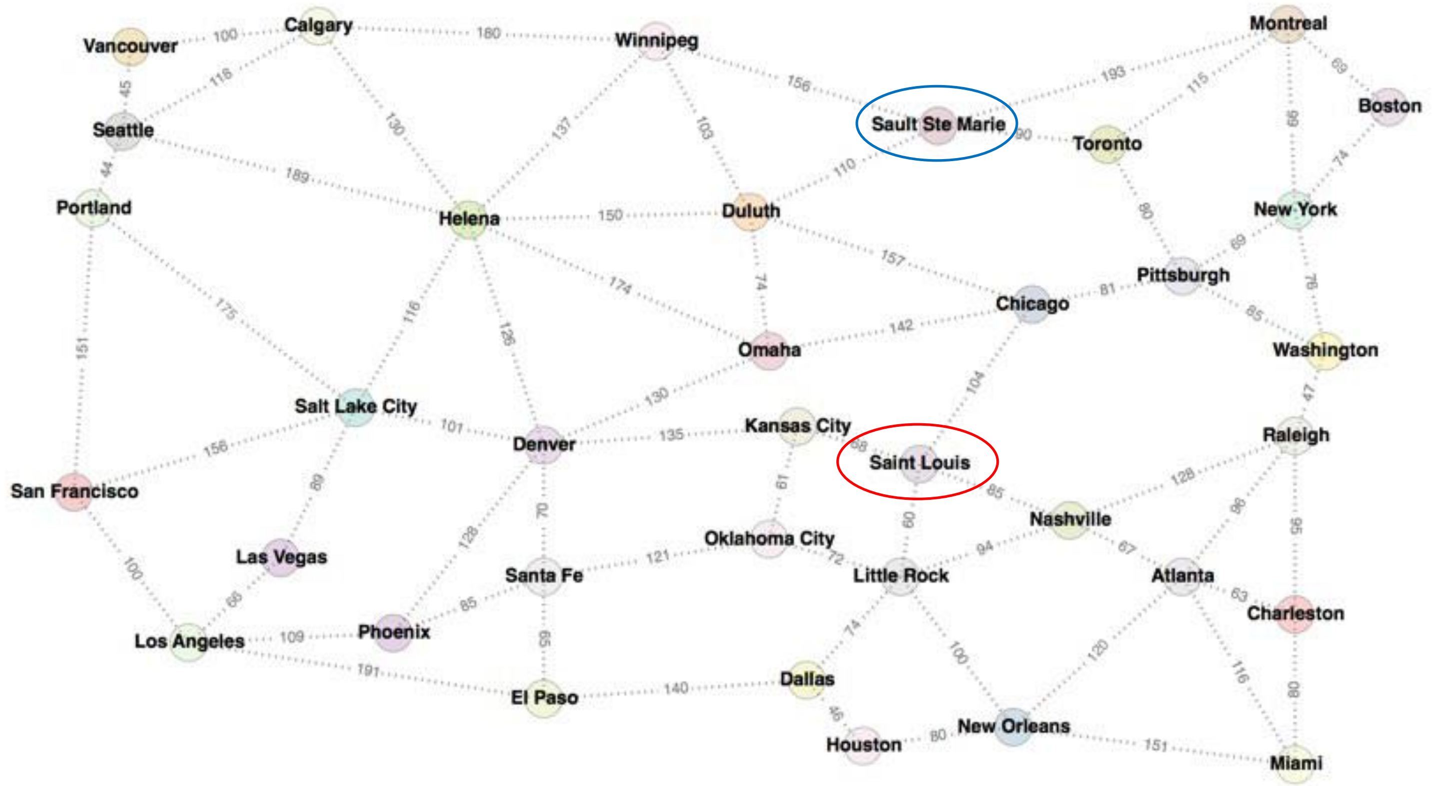
Lecture 4: Informed Search & Local Search

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



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

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.

Heuristic!

Greedy search

- Evaluation function $h(n)$ (heuristic)
- $h(n)$ estimates the cost from n to the goal
- Example: $h_{\mathrm{SLD}}(n)$ = 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
```

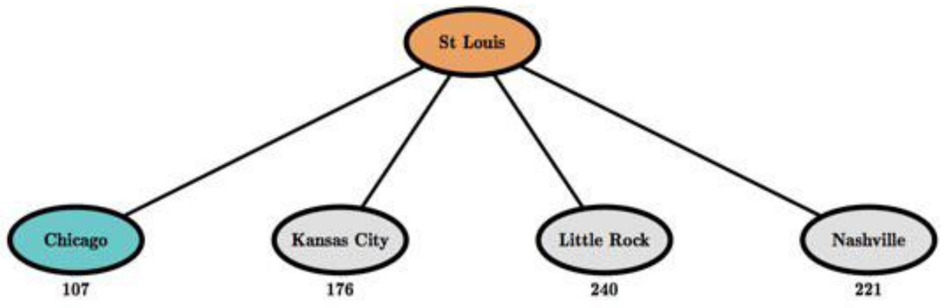
Greedy search example

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

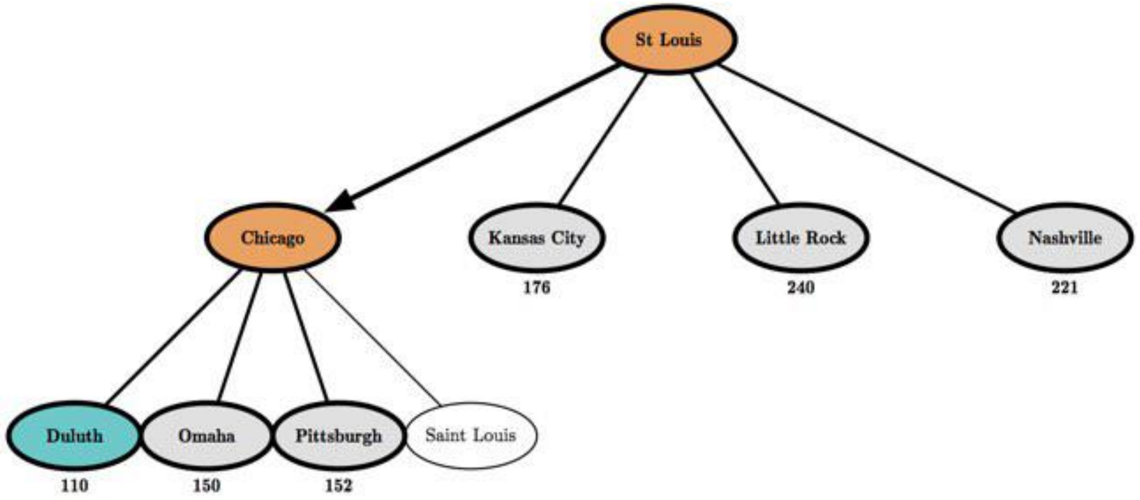
After expanding St Louis:



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

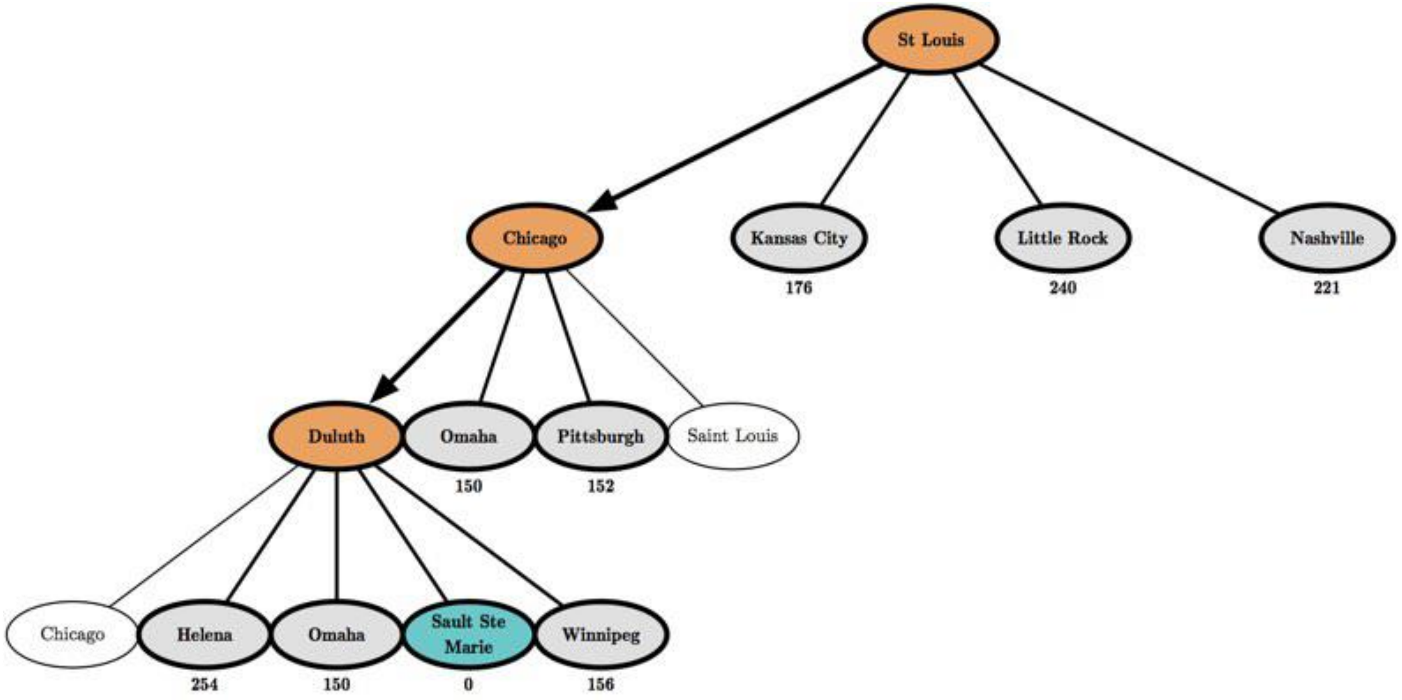
After expanding Chicago:



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:

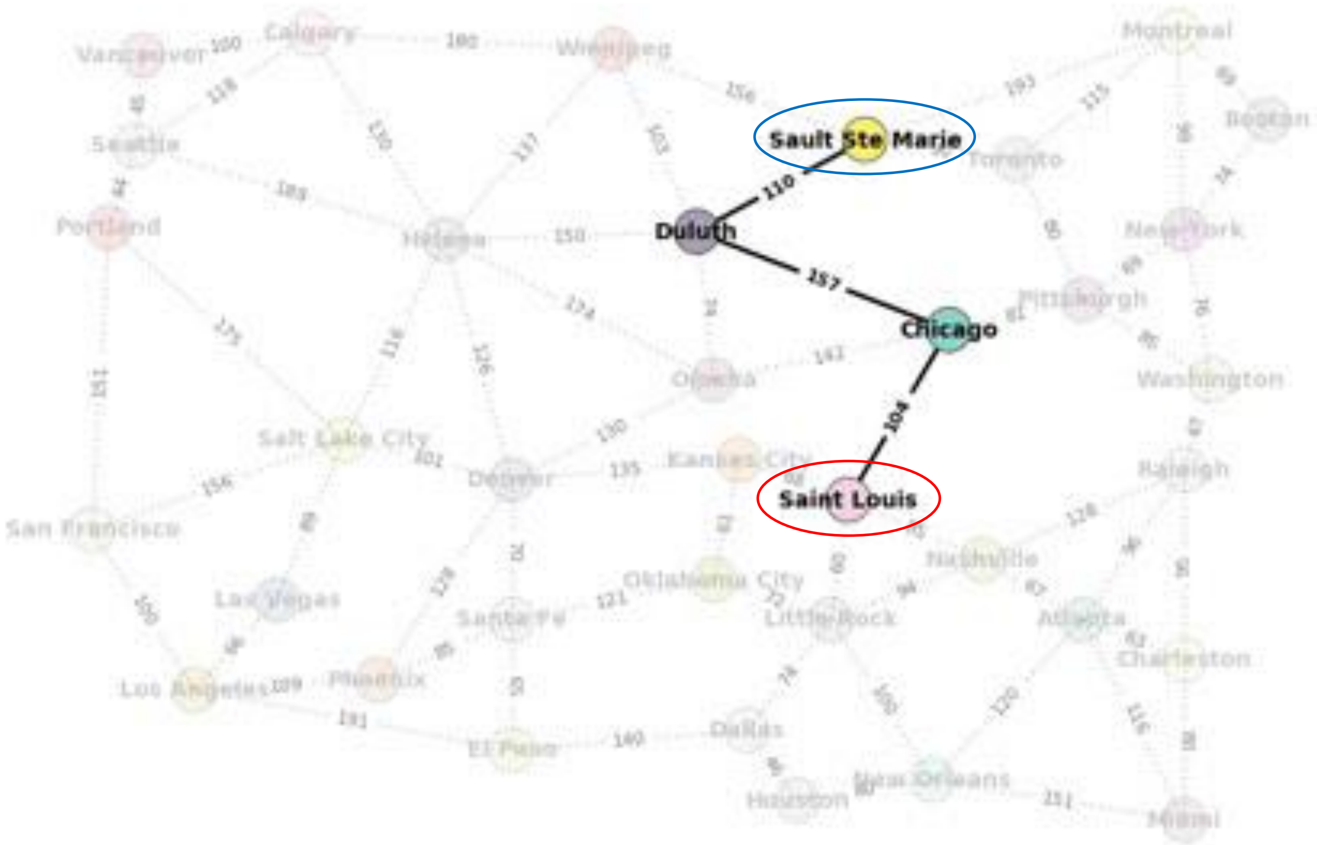


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	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

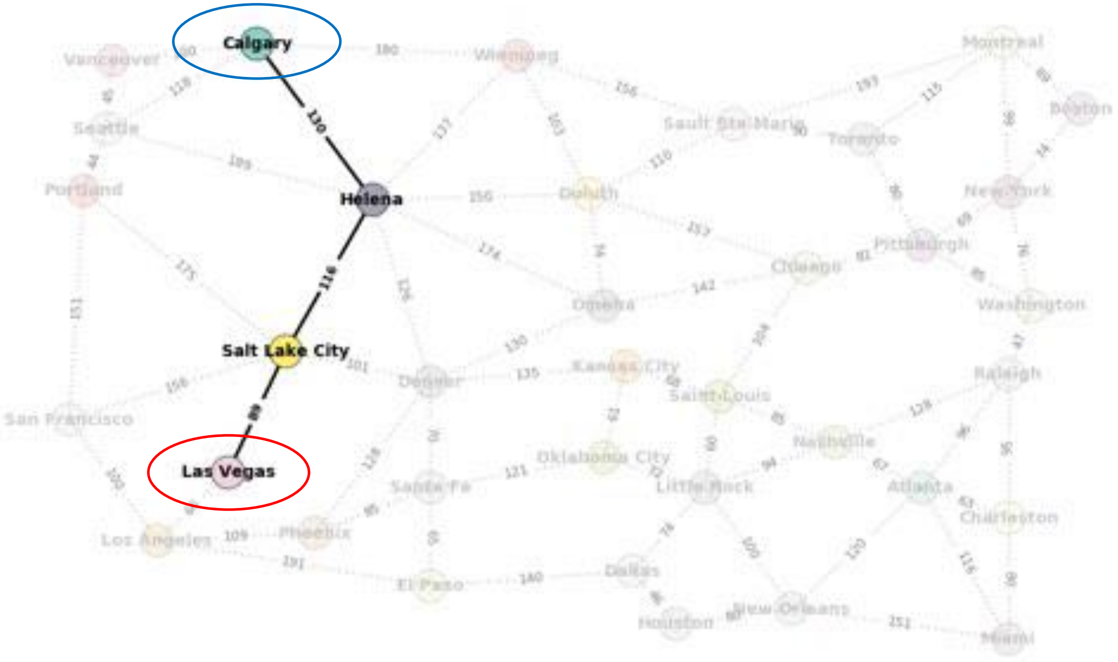
Goal: Sault Ste Marie



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 ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

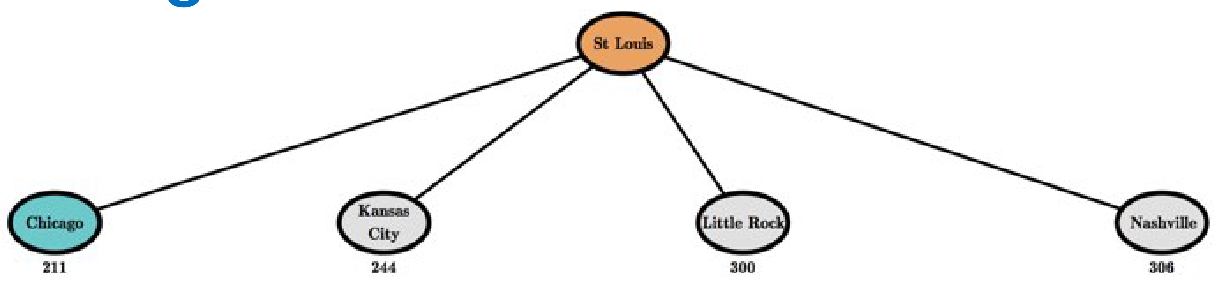
A* search example

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:

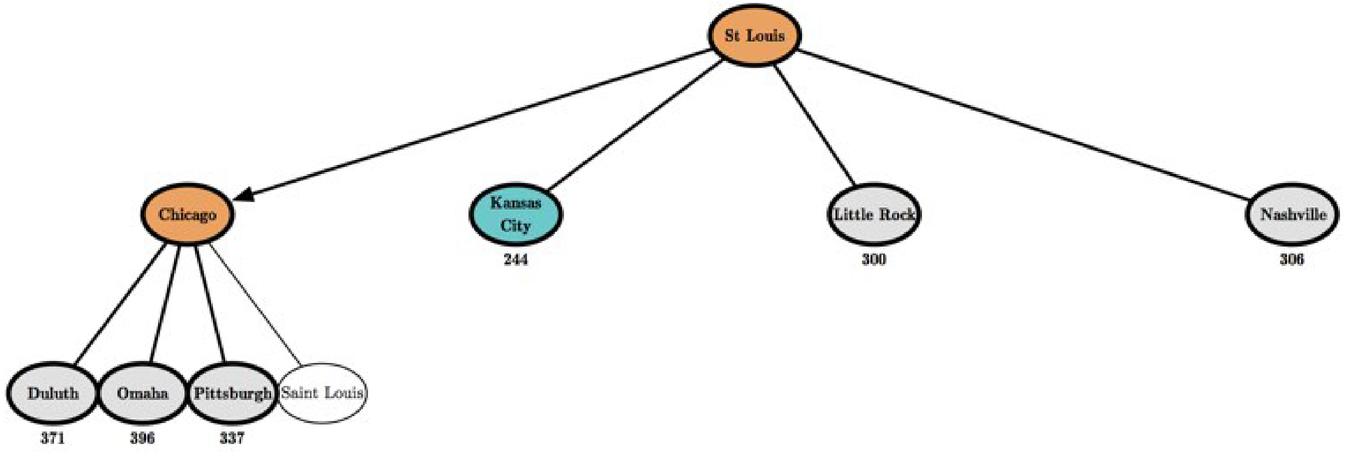


$g(n)$=104
$h(n)$=107
$f(n)$=211

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

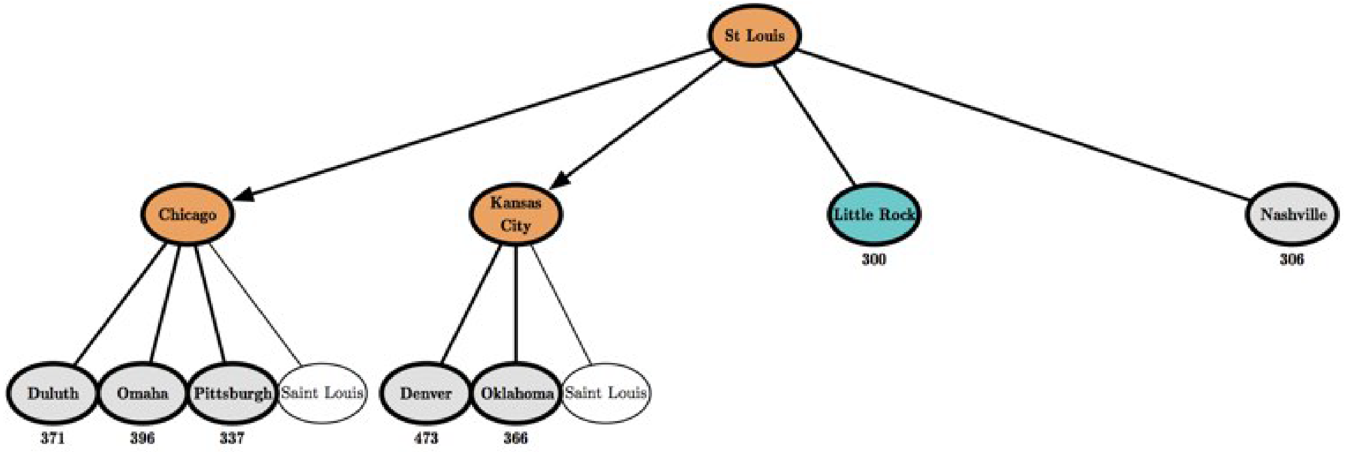
After expanding Chicago:



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

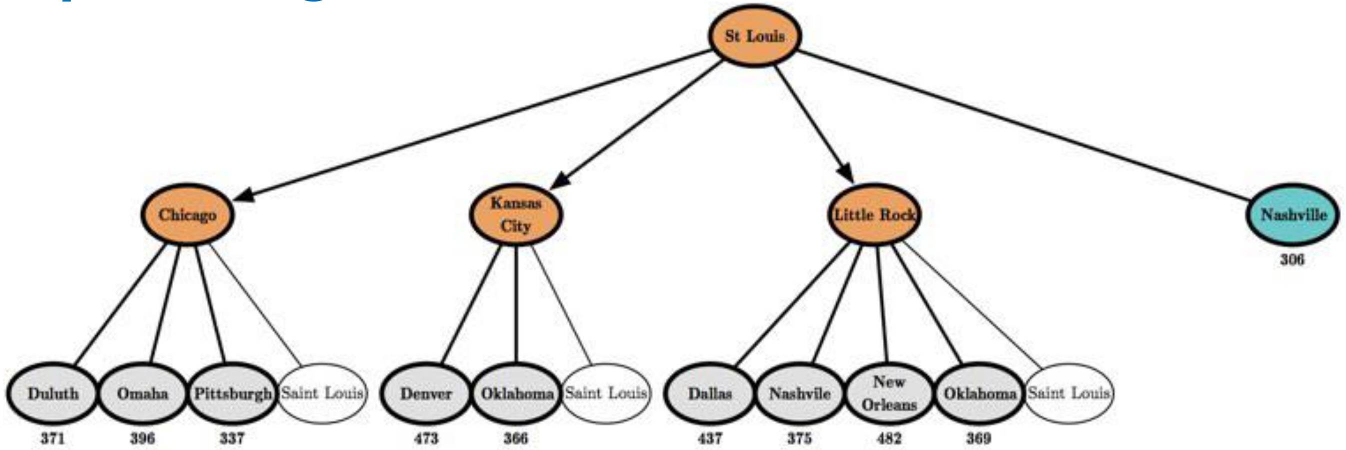
After expanding Kansas City:



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

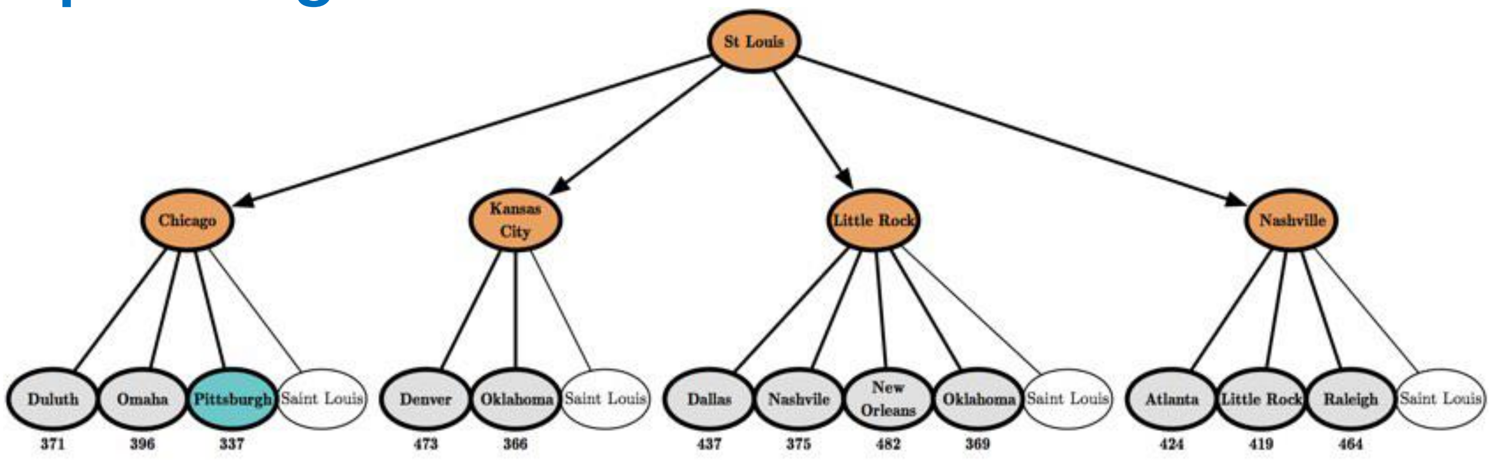
After expanding Little Rock:



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

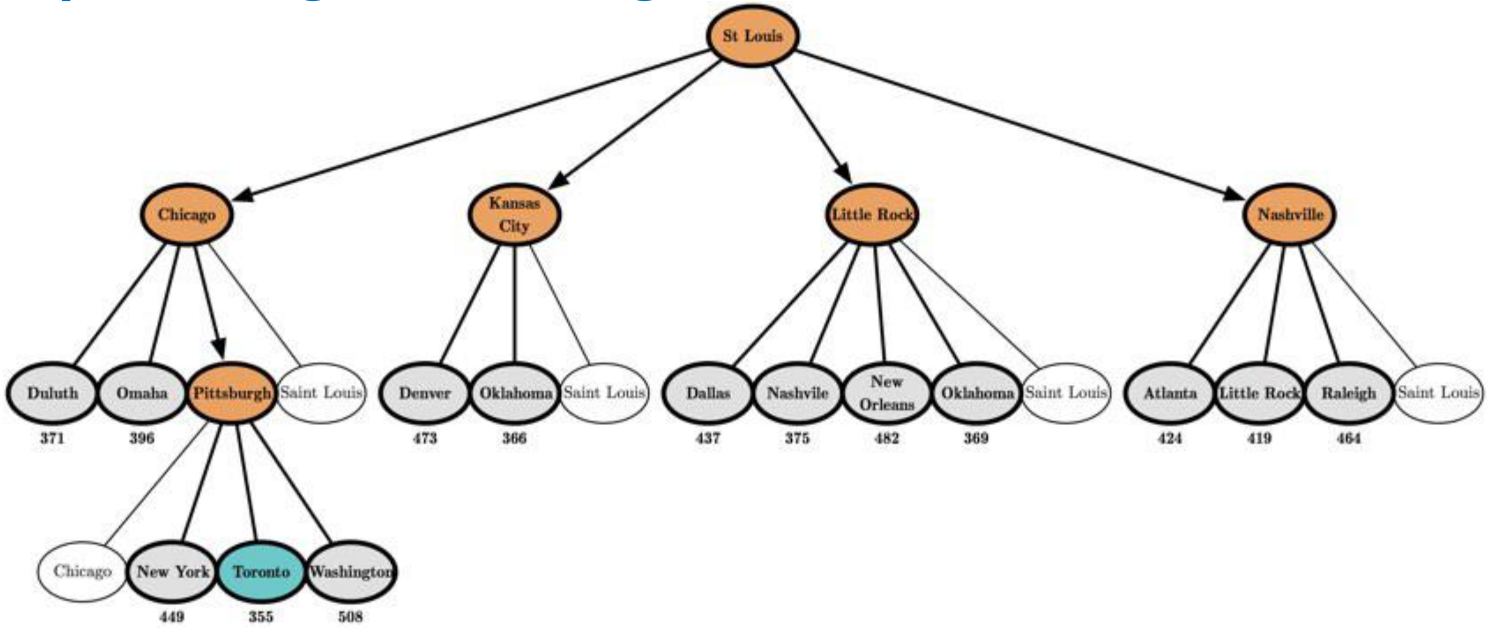
After expanding Nashville:



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

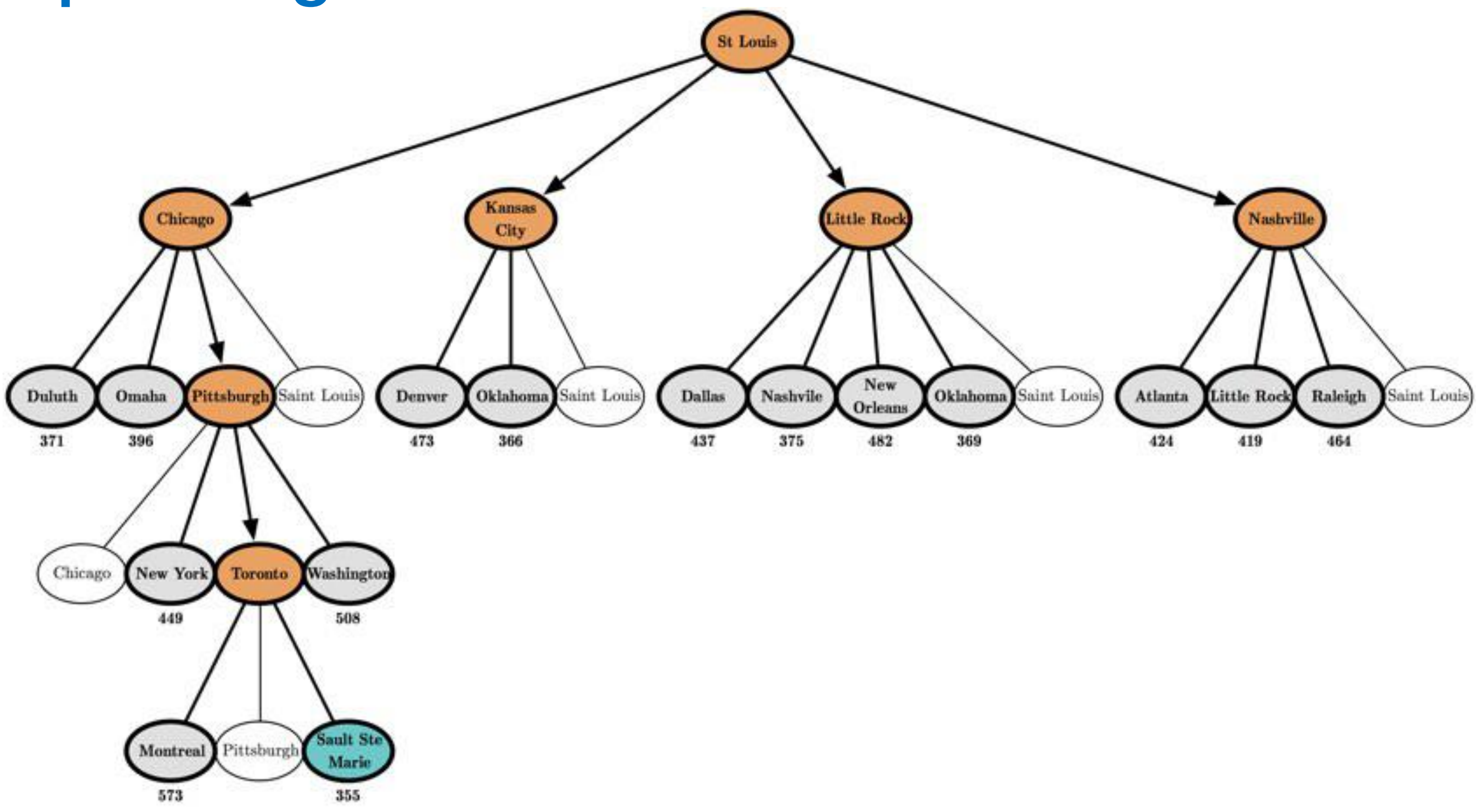
After expanding Pittsburgh:



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:

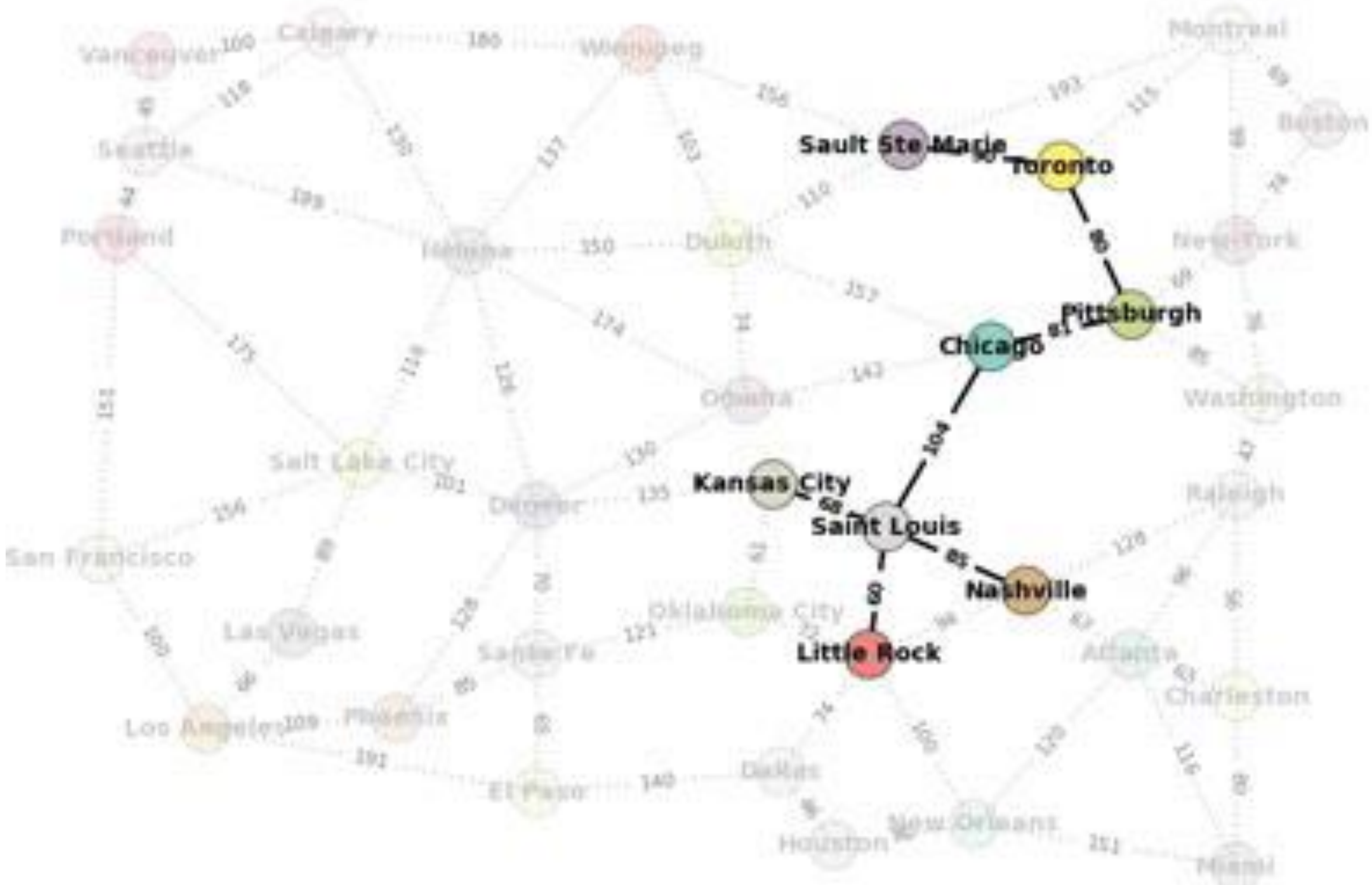


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	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

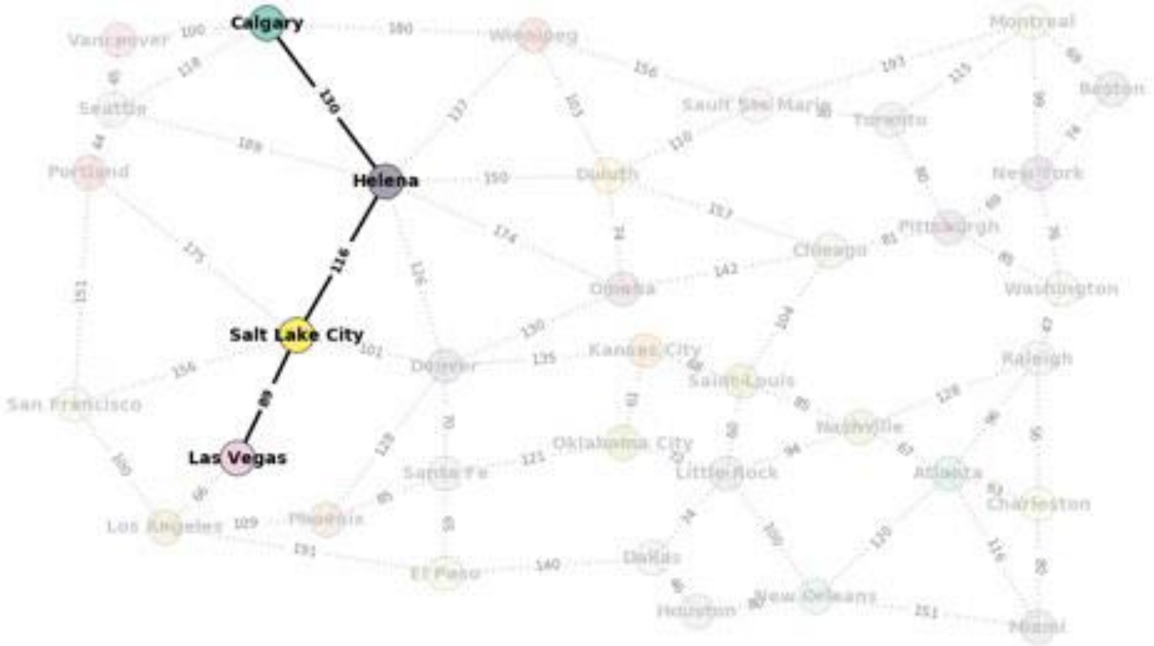
Goal: Sault Ste Marie



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 $\mathbf{f(G_s) > f(G_o)}$ **. . . (1)**
- $h(n) \le h^*(n)$ since h is admissible
  $g(n)+h(n) \le g(n)+h^*(n) = g(G_o) = f(G_o)$
  Then, $\mathbf{f(n) \le f(G_o)}$ **. . . (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.

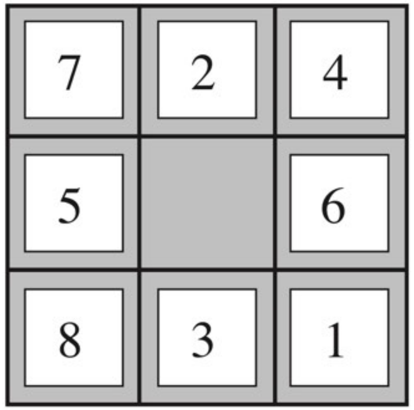
We always will go toward *Go* rather than to go *Gs*.

A* : PF Metrics

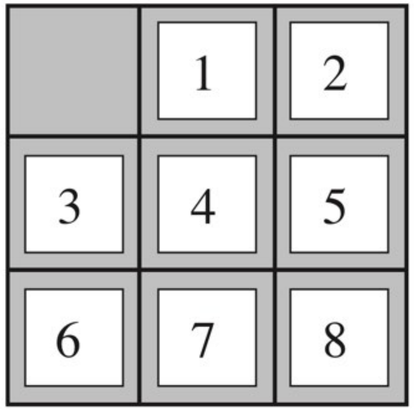
- **Complete:** Yes.
- **Time:** exponential
- **Space:** keeps every node in memory, the biggest problem
- **Optimal:** Yes!

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.



Start State



Goal State

Recap: Search Methods

- **Uniformed search**: Use **no** domain knowledge.
  - BFS, DFS, DLS, IDS, UCS
- **Informed search**: Use a heuristic function that **estimates** how close a state is **to the goal**.
  - Greedy search, A*, IDA*.

Recap: Search Methods

We can organize the algorithms into pairs where the first proceeds by layers, and the other proceeds by subtrees.

(1) Iterate on Node Depth:

- BFS searches layers of increasing node depth.
- IDS searches subtrees of increasing node depth.

(2) Iterate on Path Cost + Heuristic Function:

- A* searches layers of increasing path cost + heuristic function.
- IDA* searches subtrees of increasing path cost + heuristic function.

Recap: Search Methods

Which cost function?

- UCS searches layers of increasing path cost.
- Greedy best first search searches layers of increasing heuristic function.
- A* search searches layers of increasing path cost + heuristic function.

Local search

- Search algorithms seen so far are designed to **explore search spaces systematically.**
- Problems: observable, deterministic, known environments
- where the solution is a sequence of actions.
- Real-World problems are more complex.
- When a goal is found, the path to that goal constitutes a solution to the problem. But, depending on the applications, **the path may or may not matter**.
- If the path does not matter/systematic search is not possible, then consider another class of algorithms.

Local search

- In such cases, we can use iterative improvement algorithms, **Local search**.
- Also useful in pure **optimization problems** where the goal is to find the best state according to an **optimization function**.
- **Examples:**
  - Integrated circuit design, telecommunications network optimization, etc.
  - 8-queen: what matters is the final configuration of the puzzle, not the intermediary steps to reach it.

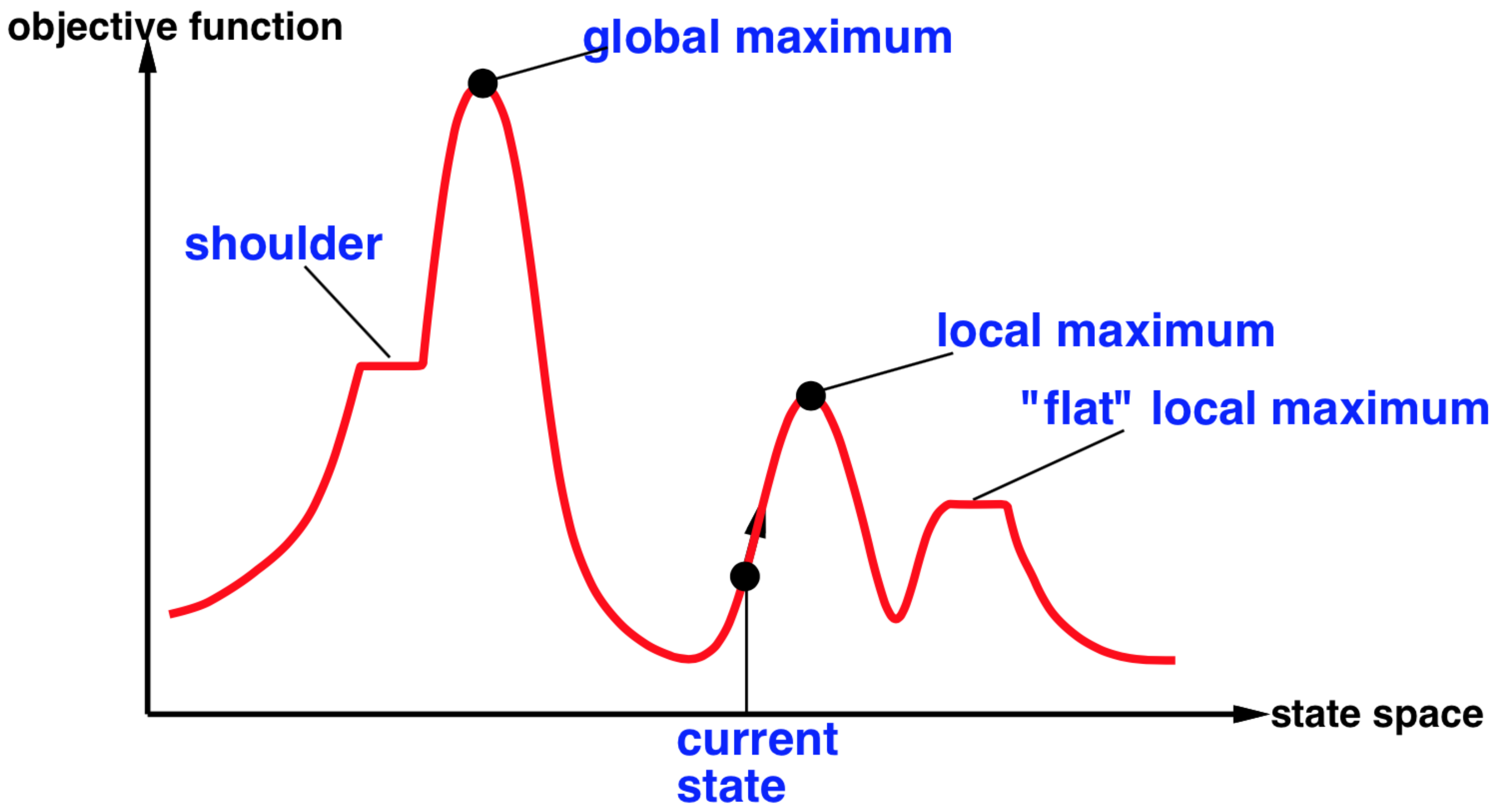
Local search

- **Idea**: keep a single “current” state, and try to improve it.
- Move only to neighbors of that node.
- **Advantages**:
  1. No need to maintain a search tree.
  2. Use very little memory.
  3. Can often find good enough solutions in continuous or large state spaces.

Local search

- **Local Search Algorithms:**
  - Hill climbing (steepest ascent/descent).
  - Simulated Annealing: inspired by statistical physics.
  - Local beam search.
  - Genetic algorithms: inspired by evolutionary biology.

Local search



State space landscape

Hill climbing

- Also called **greedy local search**.
- Looks only to immediate good neighbors and not beyond.
- Search moves uphill: moves in the direction of increasing elevation/value to find the top of the mountain.
- Terminates when it reaches a **peak**.
- Can terminate with a local maximum, global maximum or can get stuck and no progress is possible.
- A **node is a state and a value**.

Hill climbing: Pseudo-code

function HILL-CLIMBING(initialState)
    returns State that is a local maximum

    initialize current **with** initialState

    loop do
        neighbor = a highest-valued successor of current

        if neighbor.value $\leq$ current.value:
            return **current.state**

        current = neighbor

Hill climbing

Other variants of hill climbing include

- **Sideways moves** escape from a plateau where best successor has same value as the current state.
- **Random-restart** hill climbing overcomes local maxima: keep trying! (either find a goal or get several possible solution and pick the max).
- **Stochastic** hill climbing chooses at random among the uphill moves.

Hill climbing

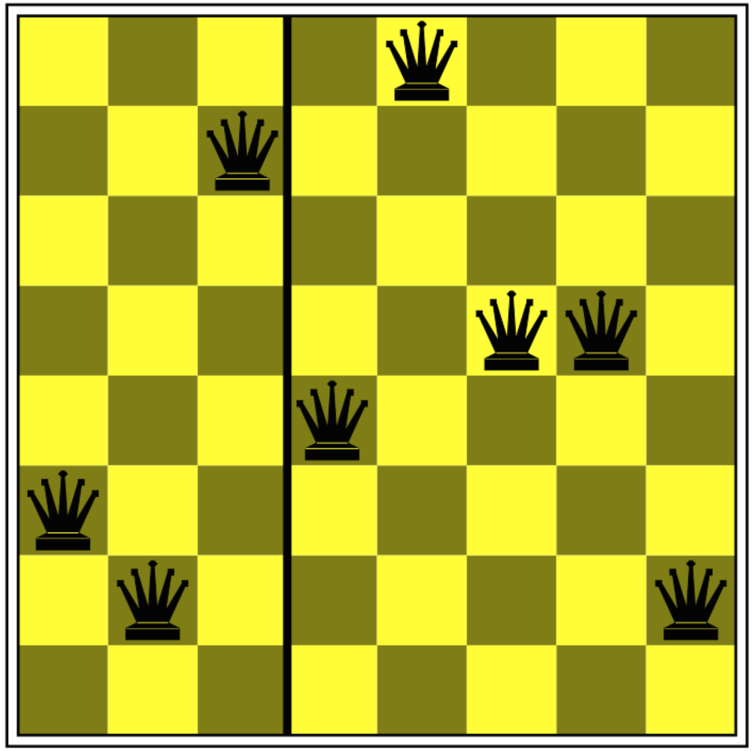
- **Hill climbing** effective in general but depends on shape of the landscape. Successful in many real-problems after a reasonable number of restarts.
- **Local beam search** maintains *k* states instead of one state. Select the *k* best successor, and useful information is passed among the states.
- **Stochastic beam search** choose *k* successors are random. Helps alleviate the problem of the states agglomerating around the same part of the state space.

Genetic algorithms

- **Genetic algorithm (GA)** is a variant of stochastic beam search.
- Successor states are generated by combining two parents rather by modifying a single state.
- The process is inspired by **natural selection**.
- Starts with *k* **randomly generated states**, called **population**. Each state is an **individual**.
- An individual is usually represented by a **string** of 0's and 1's, or digits, a finite set.
- The objective function is called **fitness function**: better states have high values of fitness function.

Genetic algorithms

- In the 8-queen problem, an individual can be represented by a **string** digits 1 to 8, that represents the position of the 8 queens in the 8 columns.



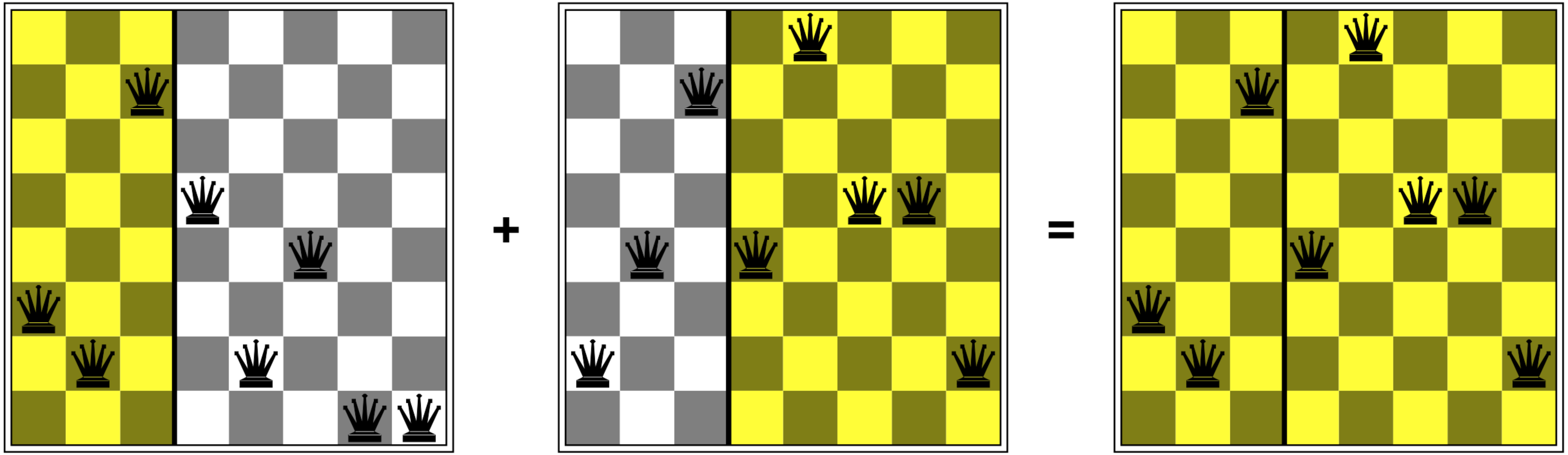
Genetic algorithms

- The objective function is called **fitness function**: better states have high values of fitness function.
- Possible fitness function is the **number of non-attacking pairs of queens.**
- Fitness function of the solution: 28.

Genetic algorithms

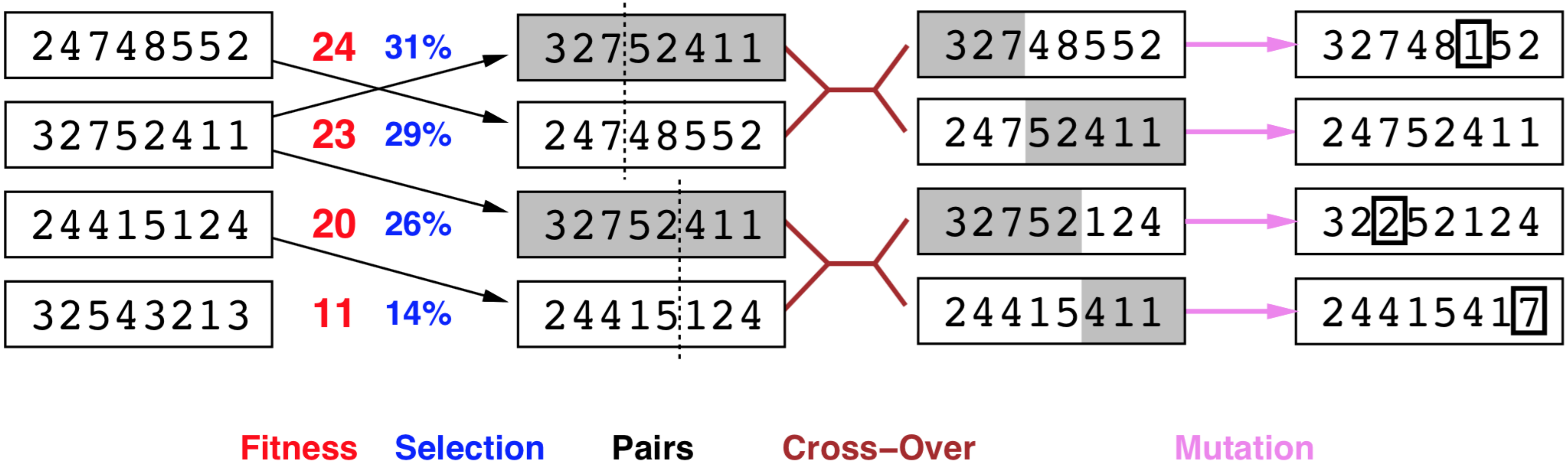
- Pairs of individuals are selected at random for **reproduction** w.r.t. some probabilities.
- A **crossover** point is chosen randomly in the string.
- **Offspring** are created by crossing the parents at the crossover point.
- Each element in the string is also subject to some **mutation** with a small probability.

Genetic algorithms



Genetic algorithms

Generate successors from pairs of states.



Genetic algorithms: Pseudo-code

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

To be continued

Simulated Annealing

- Hill Climbing → Simulated Annealing

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
    $current \leftarrow problem$.INITIAL
    while *true* **do**
        $neighbor \leftarrow$ a highest-valued successor state of *current*
        if VALUE(*neighbor*) $\leq$ VALUE(*current*) **then return** *current*
        $current \leftarrow neighbor$

Simulated Annealing

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if T = 0 **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E < 0$ **then** $current \leftarrow next$ //for maximization
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$