School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702
COMPUTATIONAL THINKING
WEEK 10: HEURISTICS

2020/21 Term 1

Heuristics Part 1: TSP

Video (25 mins): https://youtu.be/b89TO0FhKK0

Road Map

Algorithm Design and Analysis

(Weeks 1 - 5)

Fundamental Data Structures

(Weeks 6 - 9)

Computational Intractability and Heuristic Reasoning

This week ⟶

- ✦ Week 10: Heuristics
- ✦ Week 11: Limits of Computation
- ✦ Week 13: Review

The Traveling Salesman

Strategies for a computationally demanding problem

- Traveling Salesman
- Exhaustive Search
- Random Search
- Greedy Algorithm

Understanding *Complexity*

2018 Winter Olympics Torch Relay

https://en.wikipedia.org/wiki/2018_Winter_Olympics_torch_relay

- Huge undertaking
  - 80 localities in every corner of South Korea from Incheon to Pyeongchang
  - 7,500 runners covering 2,018 km
- How to compute the “optimal” route?
  - $80! = 7.2 \times 10^{118}$ possible permutations
  - With 1 Billion permutations per second, it will take 10^{100} centuries to try every one.

Bike Tour

- Suppose you decide to ride a bicycle around Ireland
  - you will start in Dublin
  - visit Cork, Galway, Limerick, and Belfast before returning to Dublin
  - minimize the number of kilometers yet make sure you visit all the cities?

	Belfast	Cork	Dublin	Galway	Limerick
Belfast	–				
Cork	425	–			
Dublin	167	257	–		
Galway	306	209	219	–	
Limerick	323	105	198	105	–

Optimal Tour

- If there are only 5 cities it's not too hard to figure out the optimal tour
  - the shortest path is most likely a "loop"
  - any path that crosses over itself will be longer than a path that travels in a big circle

Traveling Salesman Problem

- Computer scientists call the problem of finding an optimal path between n points the traveling salesman problem (TSP)
- The TSP is a famous computational problem
  - first posed by Irish mathematician W. R. Hamilton in the 19th century
  - intensely studied in operations research and other areas since 1930

This tour of 13,500 US cities was generated by an advanced algorithm that used several "tricks" to limit the number of possible tours

Required 5 "CPU-years"

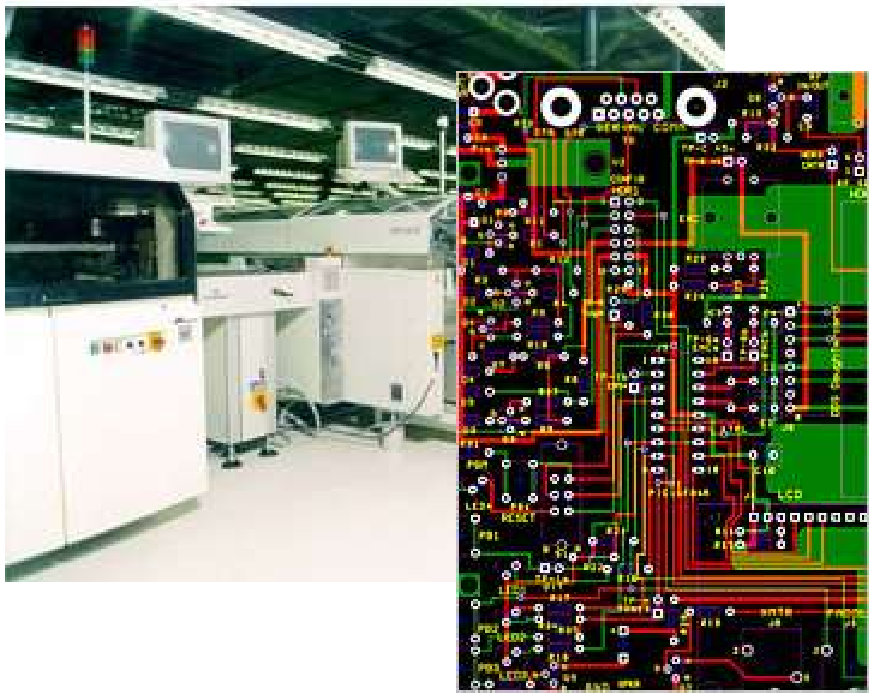
http://www.math.uwaterloo.ca/tsp/

Interesting datasets and challenges

Real-Life Applications

- The solution of several important “real-world” problems is the same as finding a tour of a large number of cities
  - Transportation and logistics : school bus routes, service calls, delivering meals, ...
  - Manufacturing: an industrial robot that drills holes in printed circuit boards
  - VLSI (microchip) layout
  - communication: planning new telecommunication networks

For many of these problems n *(the number of “cities”) can be 1,000 or more*



How Many Possible Tours?

- There is a problem with the exhaustive search strategy
  - number of possible tours of a map with n cities is $(n - 1)! / 2$
- The number of tours grows quickly as we increase the number of cities

#cities	#tours
5	12
6	60
7	360
8	2,520
9	20,160
10	181,440

The number of tours for 25 cities:

310,224,200,866,619,719,680,000

Why `(n-1)!/2` ?

- Each tour is a cycle, so it does not really matter where it begins
  - the number of cyclical tours of a map with n cities is $n!/n = (n - 1)!$
  - E.g. A-B-C-D-E is the same as B-C-D-E-A
- Assuming undirected graph, between any two cities, both directions have equal distance
  - for every tour, there is another "equivalent" tour in the opposite direction
  - E.g. A-B-C-D-E is the same as E-D-C-B-A in terms of total distance
  - divided by 2

TSP Support in GraphLab.py

- TSP graph
  - `TSPGraph()`
  - Inherits from Graph
  - `addVertex(vertex, x, y)`
    - Automatically adds an edge between all vertices and new vertex.
  - `generateRandomNodes(n)`
    - Randomly generates n vertices

Initial Strategies for Solving TSP

- Eye-balling

- Exhaustive search
  - consider all possible tours, resulting in true optimal value
  - complexity is O(`n!`)

- Random search
  - consider some number `k` of randomly chosen tours
  - complexity is O(`kn`)
  - depending on chance, the results may get close or far from the optimal value

Generalization: An Optimization Problem

- An optimization problem is associated with an objective function
  - For any solution, the objective function evaluates a *value*
  - The value indicates the goodness of this solution
  - The goal is to obtain the solution with as optimal value as possible

- Example: Traveling Salesman Problem
  - Objective function computes the distance covered by a route
  - The goal is to find a route with as short distance as possible

- What are other examples of optimization problems?

Ideal Properties of an Optimization Algorithm

- Fast
  - runs in polynomial time complexity
- General
  - works well on all instances of a problem
- Optimal
  - finds the exact optimal solution

Fast, Cheap, and Good ... Pick any two!

Trade-offs (pick any 2, but not all 3)

- General + Optimal
  - very slow
  - e.g., brute force (exhaustive search)
- Fast + Optimal
  - may not cover all cases
- Fast + General
  - Not guaranteed to find optimal solution
  - Can we find a solution that is **good enough**?
  - Yes, with heuristics strategies

Trade-offs (pick any 2, but not all 3)

- General + Optimal
  - very slow
  - e.g., brute force (exhaustive search)
- Fast + Optimal
  - may not cover all cases
- Fast + General
  - Not guaranteed to find optimal solution
  - Can we find a solution that is **good enough**?
  - Yes, with heuristics strategies

Trade-offs (pick any 2, but not all 3)

- General + Optimal
  - very slow
  - e.g., brute force (exhaustive search)
- Fast + Optimal
  - may not cover all cases
- Fast + General
  - Not guaranteed to find optimal solution
  - Can we find a solution that is **good enough**?
  - Yes, with heuristics strategies

Heuristics Techniques

- Main idea:
  - make a series of decisions in sequential steps.
  - at every step, make the best decision for the current step.
  - note that the local decisions do not guarantee best global solution.
- **Greedy algorithm**
  - construct the solution step by step, only optimize one step at a time.
- **Local search algorithm**
  - start with a feasible solution.
  - at every step, make a small modification to the solution.
- Proof of performance bound (how close we get to the optimal value) is important in the theory of computer science, but is not the focus of this course.