Nature Inspired Search and Optimisation
Advanced Aspects of Nature Inspired Search and Optimisation

Lecture 3: Optimization Problems and Local Search Algorithms

Shan He

School for Computational Science
University of Birmingham

January 21, 2020

What we learned and what we will learned

- What we learned last week:
  - Randomised algorithms
  - The important of randomness in algorithms
- We will learn this week:
  - What is optimisation
  - How to solve optimisation problems using:
    - Randomised algorithms
    - Local search
    - Stochastic local search algorithms.
- Note: Nature Inspired Optimisation and Search algorithms $\in$ Randomised algorithms, esp. Stochastic Local Search algorithm $\in$ Heuristic algorithms $\in$ Search and enumeration algorithms

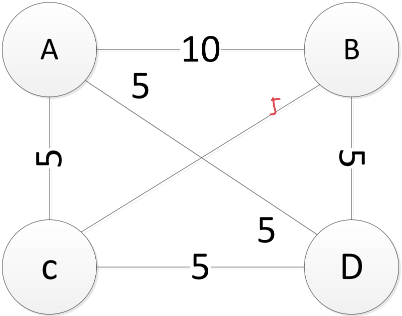
Outline of Topics

1. Travelling salesman problem
2. Solving optimisation problems
   - Solving TSP problems using heuristic algorithms
     - Randomised algorithms
     - Local search search
3. Conclusion

Travelling salesman problem (TSP)

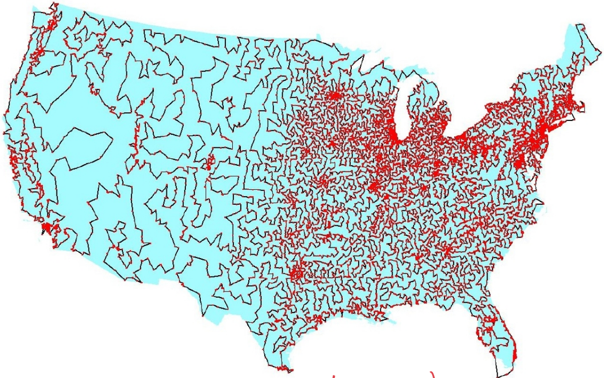
Travelling salesman problem (TSP):

- Given: a list of cities and the distances between each pair of them,
- Sought: the shortest route that visits each city exactly once and returns to the origin city



True TSP example

Optimal solution for "usa13509" from TSPLIB. Cities with population at least 500 in the continental US (in 1995). In total 13509 cities.



Connect each city | min D

Kaggle Traveling Santa 2018

Kaggle Traveling Santa 2018.

Travelling salesman problem (TSP)

- Mentioned in a travelling salesman handbook in 1832
- Formally defined by the Irish mathematician William. R. Hamilton
- One of the most prominent and widely studied combinatorial **optimisation problems** in computer science and operations research
- Many optimisation problems, e.g., Printed Circuit Boards (PSB) design can be formulated as TSP or its variations
- Conceptually simple but computationally difficult: best benchmark for development and evaluation of combinatorial **optimisation algorithms**

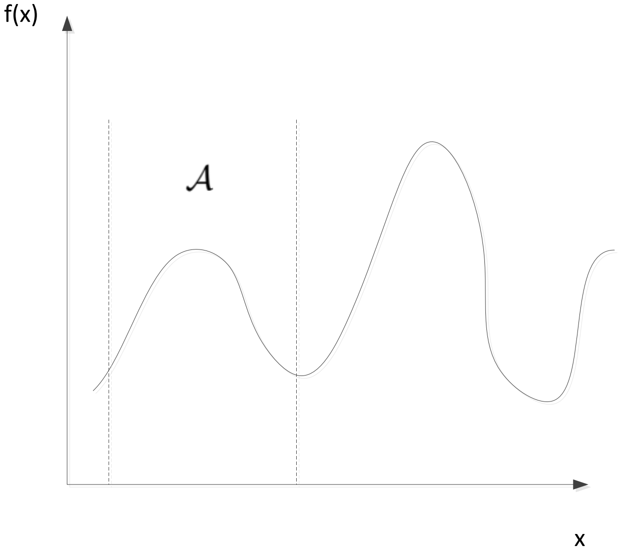
What is optimisation

- Optimization: to find the best or optimal solution to a problem
- Examples are everywhere:
  - Portfolio optimisation: you have an investment portfolio, e.g., cash and shares, you know the potential max return and financial min risk of each asset. How to build the perfect investment portfolio to maximise the return while keep risk in control? multi-objective
  - Engineering optimisation: you are required to design a product with some materials. How to design a product with minimal materials while keeping the quality?

Definition of optimisation

- Definition:
  - Given: a function $f(x)$: $\mathcal{A} \to \mathbb{R}$ from some set $\mathcal{A}$ to the real numbers
  - Sought: an element x^* in $\mathcal{A}$ such that $f(x^*) \leq f(x)$ ("minimisation") or $f(x^*) \geq f(x)$ ("maximisation") for all x in $\mathcal{A}$.
- Function $f(x)$ is called objective function, or cost function (minimisation), fitness function (maximisation and in Evolutionary Computation)
- $\mathcal{A}$ is called feasible set, which is some subset of the Euclidean space specified by a set of constraints
- The domain $\mathcal{A}$ of $f(x)$ is called the search space, while the elements of $\mathcal{A}$, e.g., $x \in \mathcal{A}$ are called candidate solutions or feasible solutions.

Definition of optimisation



Categories of optimisation problems

- Depends on the nature of objective function:
  - Linear vs non-linear
  - Linear function:
    - Additivity: $f(x+y) = f(x) + f(y)$ linear programming
    - Homogeneity: $f(\alpha x) = \alpha f(x)$ for all α
  - If it is non-linear:
    - Convex vs non-convex
    - Question: which is more difficult?
  - Multi-objective vs single objective
  - Constrained vs non-constrained
- Depends on the nature of solutions:
  - Continuous vs Discrete (also known as a combinatorial optimization)

Question

non-linear

- **Question 1:** What kind of optimisation problem is TSP?
- **Question 2:** What algorithm you can use to solve TSP?

Optimisation algorithms

- Mathematical programming algorithms, e.g., linear programming
- Search and enumeration algorithms
  - Brute force algorithms, enumerating all possible candidate solutions and check
  - Improved brute force algorithms, e.g., branch and bound algorithms
  - Heuristic algorithms within a reasonable time frame
    - Randomised algorithms
    - Local search, e.g., hill climbing and greedy search

Solving TSP

Answer to Question 2: Solve TSP using:

- Mathematical programming algorithms, e.g., linear programming
- Brute force or improved brute force algorithm, e.g., branch and bound
- **Heuristic algorithms**

Why heuristic algorithms for TSP?

- TSP is difficult for other algorithms
  - Brute force algorithms: complexity is $O(n!)$, the factorial of the number of cities.
  - Improved brute force algorithms, e.g., branch and cut algorithms: $O(1.9999^n)$ [1] $n! \longrightarrow 2^n$
    - Still time consuming, e.g., for usa13509, the running time required is 2^{13509}
    - In fact, usa13509 was solved in 1998 by Rice University using two clusters of 44 CPUs, took approximately three months from start to finish (see News)
    - The largest instance of TSP problem is an instance in TSPLIB of 85,900 cities, taking over 136 CPU-years [2]!
  - Linear programming: essentially an integer linear programming problem, itself is a NP-hard problem (See Why LP cannot solve large instances of NP-complete problems in polynomial time)

[1] Woeginger, G.J. (2003), "Exact Algorithms for NP-Hard Problems: A Survey", Combinatorial Optimization Eureka, You Shrink! Lecture notes in computer science, vol. 2570, Springer, pp. 185207. [2] Applegate, D. L.; Bixby, R. M.; Chvtal, V.; Cook, W. J. (2006), The Traveling Salesman Problem, ISBN 0-691-12993-2.

Randomised algorithms

- Two categories of randomised algorithms:
  - **Using random numbers to find a solution to a problem (today)**
  - Using random numbers to improve a solution to a problem (next lecture)
- For the first category, two representative simple algorithms:
  - Las Vegas algorithm
  - Monte Carlo algorithm ✓ no optimised target
- **Question 1**: Which one should we choose?
- **Question 2**: How to generate random solutions to TSP?

$\boxed{\text{Array}}$ random permutation

fully-connected graph

ABCD
BACD
⋮

Randomised search algoirthms

- We will demonstrate how randomised algorithm solve a small TSP problem: 48 capitals of the US (called ATT48), of which the known optimal solution by some brute force algorithms search is 10628
- Let's run the code
- Observation: Randomised search algorithms such as Monte Carlo return bad results
- **Question**: Why randomised search algorithms cannot solve TSP?

Randomised search algoirthms

- Question: Why randomised search algorithms cannot solve TSP?
- Answer: This is because the number of good solutions of TSP is just a very tiny portion of a vast number of all feasible solutions – to find a needle in a haystack vast search space
- Reminder: Heuristic algorithms
  - Randomised algorithms X
  - Local search, e.g., hill climbing and greedy search

Local search algorithms

- **Local search**: a heuristic algorithm for solving hard optimization problems
  - **Idea**: start with an initial guess at a solution and **incrementally** improve it until it is one
  - **Incremental improvement**: local changes, e.g., the algorithm iteratively moves to a **neighbour solution**
  - Search **Neighbour solution**: Depends on the definition of a neighbourhood relation on the search space, but generally based on similarity (distance) measure
  - Question: Why use neighbour solutions? What are the drawbacks?

Generic local search algorithm

Generic local search algorithm

1. x_0 := generate initial solution
   terminationflag := false
   $x := x_0$
2. while (terminationflag != true)
   Modify the current solution to a neighbour one $v \in \mathcal{A}$
   If $f(v) < f(x)$ then $x := v$
   If a termination criterion is met: terminationflag := true
   Output x

Note: termination criterion could be maximum iteration is reached or no improvement for a certain iterations.

Hill climbing algorithm

- One of the simplest local search algorithms
- Hill climbing is an algorithm that more like "climbing Everest in thick fog with amnesia"
- An iterative algorithm:
  - Starts with an arbitrary solution to a problem,
  - Iteratively searches a better solution from the current solution's **immediate neighbour solutions**
  - **Immediate neighbour solutions**: most similar solutions to the current solution.
- Two types of hill climbing:
  - 1) **Simple hill climbing**: chooses the **first** better solution
  - 2) **Steepest ascent/descent hill climbing**: compares all neighbour solutions and chooses **the best** solution – greedy search

Simple hill climbing algorithm

Simple hill climbing algorithm

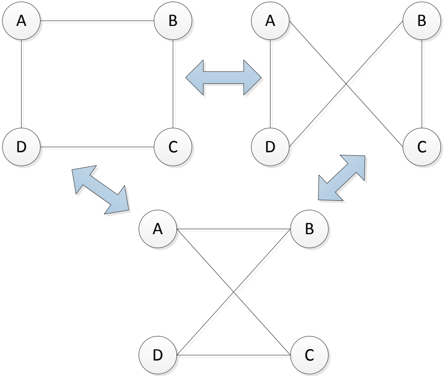
x_0 := generate initial solution
terminationflag := false
$x := x_0$
while (terminationflag != true)
  Modify the current solution to a **immediate** neighbour one $v \in \mathcal{A}$
  If $f(v) < f(x)$ then $x := v$
  If a termination criterion is met: terminationflag := true
Output x

Hill climbing for TSP problem

- **Question**: How to construct the **immediate neighbour solutions** of the current solution for TSP?

Let's take a look at some simple examples

- 2-3 cities: only one solutions
- 4 cities: 3 solutions
- **Question**: How those tours of the 4 cities TSP differ?



Conclusion

- Optimisation problems can be very difficult
- Heuristic algorithms, esp. local search are useful for difficult optimisation problems
- Optimisation is all about exploration and exploitation
- Randomness can facilitate exploration, but not exploitation (local search)