Stochastic Local Search Algorithms for TSP

Shan He

School for Computational Science
University of Birmingham

Module 06-27818 and 27819: Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

Question

Why our randomise algorithm didn't work on TSP?

Problme with randomised search algorithms

- Q: Why randomised search doesn't work on TSP??
- A: In TSP or any other optimisation problems, acceptable near optimal solutions are just a small proportion of all possible solutions
- Q: How to design algorithms to deal with this kind of problems?
- A: Most of the optimisation problems, including TSP, have a highly coherent search (solution) space, i.e., the solutions have **neighbourhood structure**.
- **Neighbourhood structure**: each solution has at least one neighbourhood solution, i.e., similar solutions
- We can exploit this coherence to design heuristic algorithms

Problme with randomised search algoirthms

- A visualisation of the search landscape of a difficult combinatorial optimisation problem: social network community detection problems

Local search algorithms

- **Local search**: a metaheuristic method 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**
  - **Neighbour solution**: Depends on the definition of a neighbourhood relation on the search space, but generally based on similarity (distance) measure

Generic local search algorithm

Generic local search algorithm

x_0 := generate initial solution
terminationflag := false
$x := x_0$
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:
  - **Simple hill climbing**: chooses the **first** better solution
  - **Steepest ascent hill climbing**: compares all neighbour solutions and chooses **the best** solution

Simple hill climbing algorithm

Simple hill climbing algorithm

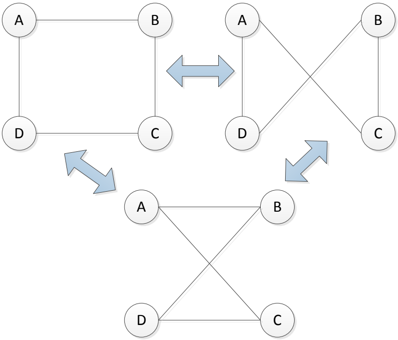
$x_0 :=$ generate initial solution
terminationflag := false
$x := x_0$
while (terminationflag != true)
    Modify the current solution to a **immediate** neighbour one $v \in \mathcal{A}$
    If $f(v) < f(x)$ then $x := v$
    If a termination criterion is met: terminationflag := true
Output x

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?



2-Opt algorithm

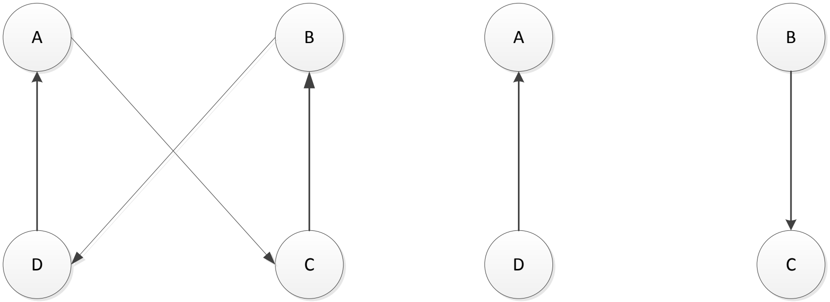
- 2-Opt: A simple local search operator first proposed by Croes in 1958 [1] for solving the travelling salesman problem.
- Basic idea: **two immediate neighbour solutions can be two routes (cycles) only differ from two edges**
- Swapping two edges can result in two **immediate** neighbour solutions
- Detailed swapping steps:
  - Step 1: removal of two edges from the current route, which results in two parts of the route.
  - Step 2: reconnect by two other edges to obtain a new solution

[1] G.A. Croes. A method for solving traveling-salesman problems. Operations Research. 1958

2-Opt algorithm

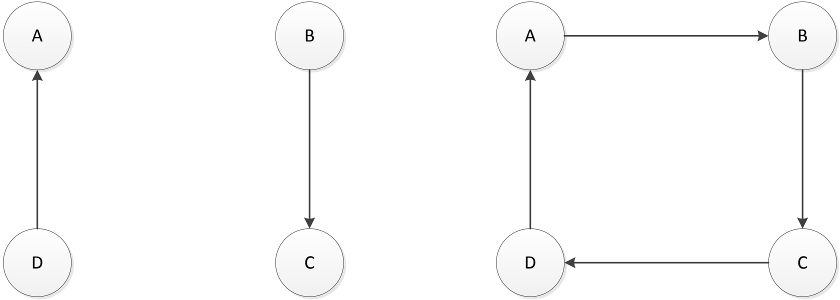
Suppose we have a route: A $\longrightarrow$ C $\longrightarrow$ B $\longrightarrow$ D $\longrightarrow$ A, which is obviously not optimal. Let's see how to swap:

Step 1: removal of two edges from the current route, which results in two parts of the route



2-Opt algorithm

Step 2: reconnect by two other edges to obtain a new solution.



This will result in an optimal route: $A \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow A$

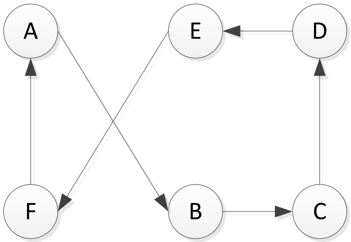
How to implement 2-Opt algorithm?

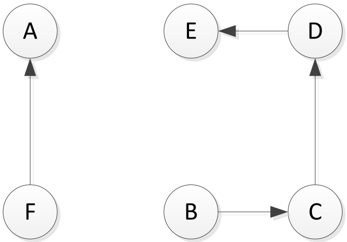
- Compare the two solutions:
  - A $\longrightarrow$ C $\longrightarrow$ B $\longrightarrow$ D $\longrightarrow$ A
  - A $\longrightarrow$ B $\longrightarrow$ C $\longrightarrow$ D $\longrightarrow$ A
- We observed that we swap two **adjacent** cities, e.g., B and C in a route can create two **immediate** neighbour solutions
- But how about this 6 cities TSP problem?

2-Opt algorithm

Suppose we have a route: A $\longrightarrow$ B $\longrightarrow$ C $\longrightarrow$ D $\longrightarrow$ E $\longrightarrow$ F $\longrightarrow$ A, which is obviously not optimal. Let's see how to swap:

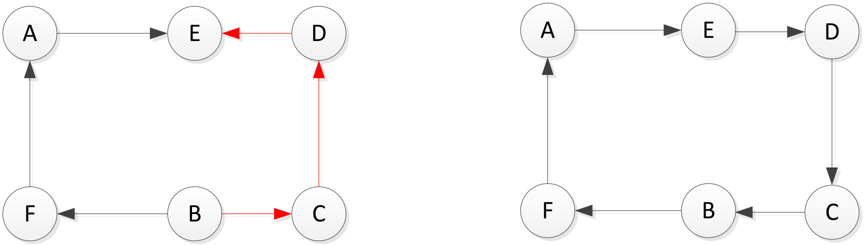
Step 1: removal of two edges from the current route, which results in two parts of the tour





2-Opt algorithm

Step 2: reconnect by two other edges to obtain a new solution



We need to reverse the order of B $\Leftarrow$ C $\Leftarrow$ D $\Leftarrow$ E in order to get A $\longrightarrow$ E $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ B $\longrightarrow$ F $\longrightarrow$ A

2-Opt algorithm: implementation

2-Opt algorithm

$route$:= initial TSP solution
i,j := two cities for swapping
Step 1: take $route[1]$ to $route[i-1]$ and add them in order to $newroute$
Step 2: take $route[i]$ to $route[k]$ and add them in reverse order to $newroute$
Step 3: take $route[k+1]$ to end and add them in order to new $newroute$
Output $newroute$

Exercise: Simple hill climbing for TSP

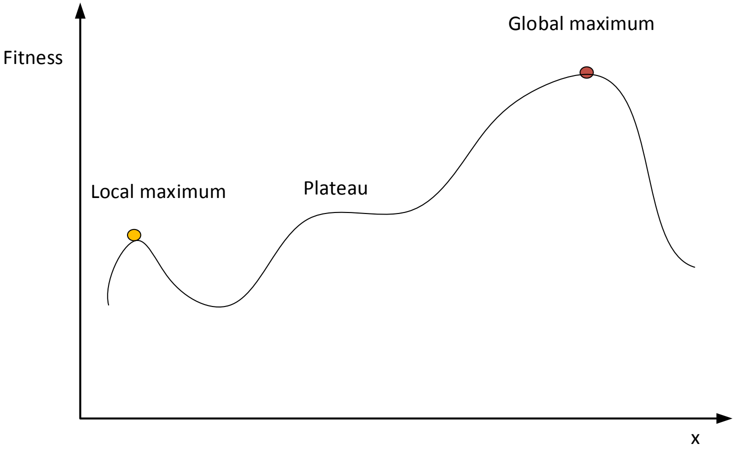
- Implement the 2-Opt algorithm by editing my empty twoopt.m file
- First test your implementation using
```
test_city = ['A', 'B', 'C', 'D', 'E', 'F'];
newroute = twoopt(test_city, 2, 5)
```
- Once you validate your 2-opt algorithm, then execute:
```
clear all
load('cities.mat')
[dist route] = simple_hill_climbing_two_opt(cities)
```
- Execute
```
[dist route] = simple_hill_climbing_two_opt(cities)
```
a few times, what is your observation?
- Modify the simple hill climbing algorithm as the steepest ascent/descent hill climbing algorithm.

Issues about hill climbing

- We can observe: the algorithm got stuck at an improved solution (local optimum) but still not the best (global optimum)
- Global vs local optimum
  - **Local optimum**: a solution that is optimal (either maximal or minimal) within a neighbouring set of candidate solutions
  - **Global optimum**: the optimal solution among all possible solutions
- General issues with local search:
  - Find only local optima unless the search space is unimodal (only one optima)
  - Even for unimodal problem, if there exists plateau, hill climbing might not perform well.

Fitness landscape with global/local optima

Fitness landscape of a 1-dimensional optimisation problem



Intensification vs Diversification

- **Intensification** (Exploitation): search small region around the current solution
    - Aim to improve a promising solution S that we already have at hand by searching in the vicinity of S
    - Local search $\longrightarrow$ local optimum
- **Diversification** (Exploration): search large unknown region of the search space
    - Aim to find other promising solutions that are yet to be refined
    - Need to escape from current local optimum $\longrightarrow$ Randomness can help
    - Global search $\longrightarrow$ global optimum

Randomised search vs Local search

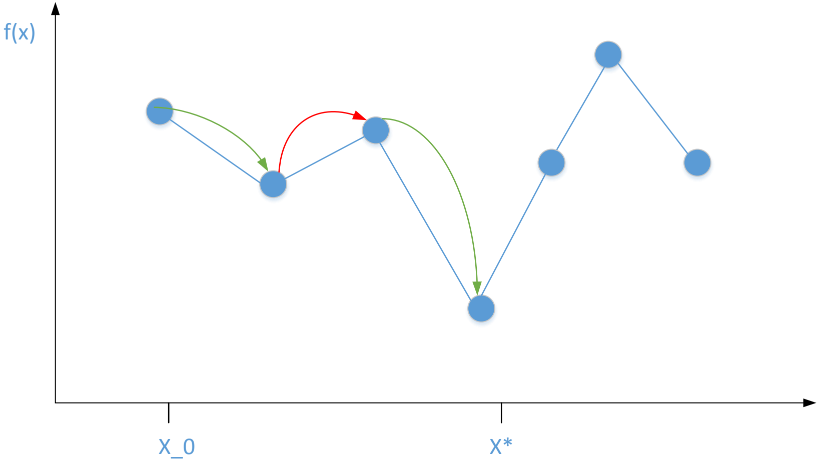
- **Randomised search**:
  - Good at exploration, e.g., to search large unknown region of the search space
  - Not good at exploitation, e.g., to search small region around the current solution
  - Especially bad for problems where good solutions are just a small portion of all possible solutions
- **Local search**:
  - Good at exploitation: capable to find local optimum
  - Not good at exploration: gets stuck into local optimum
- Question: can we combine these two heuristics?

Stochastic local search: Main idea

- Main idea: escape or avoid (more difficult) local optima
- How: Introduce randomness into local search algorithm to escape from local optima
- Escape strategies:
  - **Random restart**: simply restart the local search from a random initial solution
    - Applicable when:
      1. Number of local optima is small
      2. The cost for restarting the local search is cheap
  - **Perform random non-improving step**: randomly move to a less fit neighbour

Search trajectory of stochastic local search

Minimisation problem, e.g., TSP problem.



Stochastic Simple hill climbing algorithm

Stochastic 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 $f(v) < f(x)$ or $rand(1) < P$ then $x := v$
    If a termination criterion is met: terminationflag := true
Output x

Note: P is a small probability.

Exercises: Stocastic simple hill climbing for TSP

- Modify the Simple Hill Climbing Search Algorithm code to construct a Stochastic Hill Climbing Search Algorithm.
    - Hint 1: instead of rejecting worse solutions (immediate neighbours), accept them with a very small probability, e.g., 0.001.
    - You also need to specify the maximum iteration, otherwise your algorithm will never terminate

Simulated Annealing (Minimisation, e.g., TSP problem)

- Main idea: Escape from local optima with random non-improving step:
  - Accepting good solution with the probability of 1, e.g., $P := 1$ if $f(v) < f(x_t)$
  - Accepting worse solution with a certain probability, e.g., $P := e^{-\frac{f(v)-f(x_t)}{T(t)}}$ if $f(v) \geq f(x_t)$
- The annealing schedule $T(t)$ will slowly decrease in the probability of accepting worse solutions
- You can implement your SA from the Stochastic simple hill climbing algorithm

Generic Simulated Annealing algorithm for minimisation

$x := x_0$; $e := f(x)$ // Initial solution, objective function value (energy).
$x_{best} := x$; $x_{best} := x$ // Initial "best" solution
$k := 0$ // Count evaluation number.
while $(k < k_{max})$
    $T := temperature(t_0)$ // Temperature calculation.
    $x_{new} := neighbour(x)$ // Pick some neighbour.
    $e_{new} := f(x_{new})$ // Compute its objective function value.
    if $P(e, e_{new}, T) > R(0, 1)$ then // Should we move to it?
        $x := x_{new}$; $e := e_{new}$ // Yes, change state.
    if $e_{new} < e_{best}$ then // Is this a new best?
        $x_{best} := x_{new}$; $e_{best} := e_{new}$ // Save as 'best found'.
    $k := k + 1$ // Increase Evaluation
Output x_{best}

$P := 1$ if $e_{new} < e$, and
$P := e^{\frac{e - e_{new}}{T}}$ otherwise
Annealing schedule $temperature()$ defines how to decreased temperature from a initial temperature t_0.