Example: $f \rightarrow -f' = -13$ meaning that the new point f' is better than f.

Probability p of acceptance as a function of T for the case where the new v_n is 13 points better than the current solution v_c.

T	$e^{\frac{-13}{T}}$	p
1	0.000002	1.00
5	0.0743	0.93
10	0.2725	0.78
20	0.52	0.66
50	0.77	0.56
10^{10}	0.999	0.5

The greater the value of T, the smaller the importance of the relative merit of the competing points v_c and v_n.

If T is huge (i.e. $T = 10^{10}$) the probability of acceptance approaches 0.5. The search becomes random.

If T is very small (i.e $T = 1$) the stochastic hill-climber reverts into an ordinary hill-climber.

. . .

If the new point has the same ment as the current point ($f' = f$) the probability of acceptance is @.5.
→ It doesn't matter which one you chose because each are of equal quality.

- Should the value of T be adjusted dynamically?
  → No. It must remain constant.

How do we determine the value of T if the values of v_c and v_n keep changing with the problem?

- Make it exactly half of the ranges between $p = 0.5$ and $p = 1.0$.
  - We are looking at about $p = 0.75$ which is in between random search and an ordinary hill climber.

- What if you run a while loop to try all the possible values of T until you find the most optimal one for the current problem?

$$p = \frac{1}{1 + e^{\left(\frac{f' - f}{T}\right)}}$$ → if $f' - f = 0$ we want the choice to be random

$$\therefore \quad p = \frac{1}{1 + e^{(0/20)}} = \boxed{p = 0.5}$$ Q.E.D.?

$$\underline{T = 20}$$

- Should the SHC have a random restart?

The value of T is entirely dependant on the dataset being used.

- Take the first tour as the sample fitness to determine the value of T which yields $p = 0.5$ when v_n and v_c are equal to 0.

"T is dependant on the datasets size"
→ But what is the relationship?

$$p = \frac{1}{1 + e^{\frac{(148483.09 - 150274.74)}{(T = 48.0)}}}$$

$$= \frac{1}{1 + e^{\frac{(-1791.65)}{48}}}$$

$$= \frac{1}{1 + e^{(-37.33)}}$$

$$= \frac{1}{1 + 6.13461 \times 10^{-17}}$$

$$= 1.0$$

However, this T value does not produce $p = 0.5$ when $f' - f = 0.0$.

How to write a function that will automatically determine the value of T?

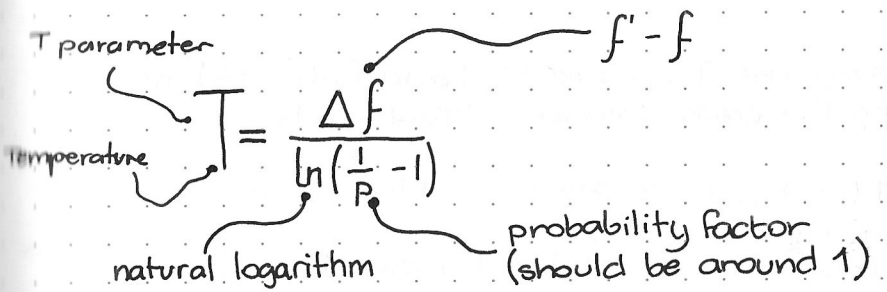
$$p = \frac{1}{1 + e^{(\Delta f / T)}} \qquad 1 + e^{(\Delta f/T)} = \frac{1}{p} \qquad e^{\Delta f/T} = \frac{1}{p} - 1$$

$$\frac{\Delta f}{T} = \ln\left(\frac{1}{p} - 1\right) \qquad \frac{1}{p} - 1 = e^{-1}$$

$$e^{-1} = \frac{1}{p} - 1$$

$$\frac{1}{e^{-1} + 1} \qquad T = \frac{\Delta f}{\ln\left(\frac{1}{p} - 1\right)}$$

Where $p = 0.03$



- Let the algorithm accept worse fitness function values during its search.
- Probability of accepting the chance is based on how big the Δf is.

```
if the value is worse than old value {
    choose if to accept it based on p {
        ...
        ...
    }
else use the better value.
```

- Where within the algorithm do we determine the value of T given that we want to get Δf for calculations?

Isn't the number of iterations = (no of cities!)?

- The greater the value of T, the smaller the relative importance of the competing points.

In theory the T parameter (and T_0 for SA) need setting for each dataset. (Tedious!)

1. T (or T_0) is related to the fitness.
2. The fitness is related to the distance matrix.

$\therefore \quad T = F(D) = Sum(D) / K$

F is the sum of all of the values in D (a distance matrix for the instance TSP)

K is constant.

→ find K. (1000.0 start)

1. Calculate the optimal temperature for the dataset using $T = F(D) = Sum(D) / K$.
2. Use that temperature for the SHC algorithm.

How do we calculate K in order to calculate T in order to calculate P?

- Use a hashmap to store values of K and fitness.

Map < Integer, Double >

K value — fitnessScore

The next stage is to iterate through the hashmap to find the lowest value and find its key to be used as K in the calculation $T = F(D) / K$.

→ This could be simplified by using an array of doubles. (X) ← IGNORE THIS SHIT. HE'S CRAZY!

This does mean, however, that you end up running nested for loops which are 3 levels deep.

Remember that this calculation need only be ran once when the new dataset is to be loaded.
→ Perhaps lower the number of iterations for scoring the K?
→ I wonder if RMHC would have been sufficient for this? But then how would we know if T is effective?

Currently it takes way too long to calculate the value of K $\therefore$ T. It might be beneficial to experiment with different numbers of iterations to achieve a better balance. (Tweak the numbers)

Number of iterations for RRHC = $\frac{\text{Number of iterations}}{\text{No of it. for RRHC}}$ 10

Number of times for RRHC = $\frac{\text{No of it. for RRHC}}{10}$

Calculating the fitness of the optimal tour:

```
Utilities.FitnessFunction(tour);
```

Simulated Annealing

- an attempt to improve the Hill Climbing algorithm
- it allows a worse solution to be accepted so that local maximums can be circumnavigated
  → under certain conditions - similar to SHC
- "annealing" refers to the fact that the chance of accepting a worse solution reduces as the algorithm progresses.
- the annealing process is simulated through maintaining a decreasing temperature.
- the temperature should have reached zero at the end of the algorithms run.

  → parameters T_0 and λ need defining, along with the acceptance function PR.

- if ITER is known (user defined number of iterations) then λ can be calculated.

  → For T_{ITER} we choose a small value greater than 0. (e.g. 0.001)

$$T_{i+1} = \lambda T_i$$

$$T_1 = \lambda T_0$$

$$T_2 = \lambda T_1 = \lambda\lambda T_0 = \lambda^2 T_0$$

$$T_3 = \lambda T_2 = \lambda\lambda T_1 = \lambda\lambda\lambda T_0 = \lambda^3 T$$

$$T_{i+1} = \lambda^{i+1} T_0$$

$$T_{ITER} = \lambda^{ITER} T_0$$

$$\lambda^{ITER} = \frac{T_{ITER}}{T_0}$$

$$\lambda = \sqrt[ITER]{\frac{T_{ITER}}{T_0}}$$

- this way we only have two parameters to determine: T_0, ITER
- ITER can be selected by trial and error, the algorithm is very fast so we can select a large value and reduce it if necessary.
- determining T_0 can be a problem:
  → too small and the algorithm will behave like a hill climber
  → too large and the algorithm may never converge

- The acceptance function PR is defined as follows:

$$PR(f', f, T_i) = \exp\left(\frac{-\Delta f}{T_i}\right)$$

Where $\Delta f = |f' - f|$

$|x|$ means the absolute value of x, and equals $-x$ if $x \leq 0$ or x if $x \geq 0$.

SIMULATED ANNEALING ALGORITHM

INPUT: T_0 Starting temperature
Iter number of iterations
λ the cooling rate

```
1) Let x = a random solution
2) For i = 0 to Iter - 1
3)     Let f = fitness of x
4)     Make a small change to x to make x'
5)     Let f' = fitness of new point x'
6)     If f' is worse than f Then
7)         Let p = PR(f', f, T_i)
8)         If p < UR(0,1) Then
9)             Reject change (keep x and f)
10)        Else
11)            Accept change (keep x' and f')
12)        End If
13)    Else
14)        Let x = x'
15)    End If
16)    Let T_{i+1} = λT_i
17) End For
```

OUTPUT: The solution x

cooling rate = 0.003

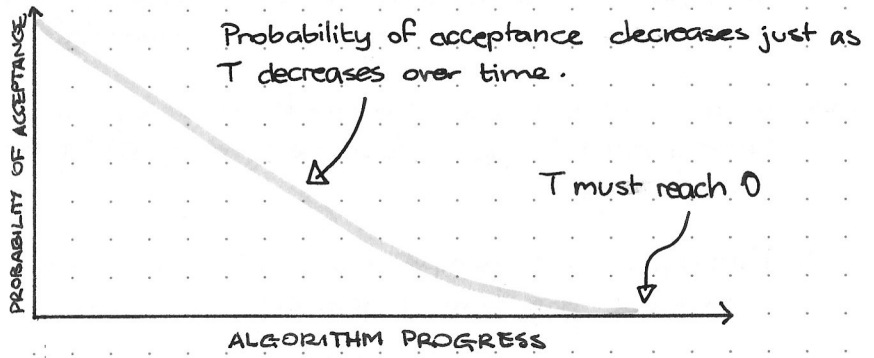
Given the implemented SA code, how do we calculate the temperature array and cooling rate?

$$\text{cooling rate } (\lambda) = \sqrt[\text{ITER}]{\frac{T_{\text{ITER}}}{T_0}}$$

Read from the array with

T_{ITER} = small value greater than 0.

→ After implementation, SA reports worse fitness than RMHC. Need to check T and λ calculations. (More research required)



How do we calculate T_0? There must be a method of calculating the value based on data given.

The notion of slow cooling implemented in SA is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is explored.

For a large T, the evolution is sensitive to coarser energy variations, whereas small T is sensitive to finer energy variations.

Initially T is set to a high value and is decreased at each step following some annealing schedule.

The errors in the SHC and SA might stem from an incorrect temperature calculation.

→ The method is used in both algorithms which determine the starting temperature of the stochastic function. Therefore setting of this value inevitably affects the overall performance of both algorithms.

→ Since the temperature search utilises the SHC it inevitably has a chain effect. Is there a relationship between temperature and fitness which I am bluntly missing?

→ Is this value being shifted too often? What would happen if it were only calculated once?

→ Perhaps the temperature value is not being changed the way it should?

[ ] start temp —(decreased by cooldown)→ 0.0

no. of iterations

↳ decreased within each loop.

$temp_{i+1} = temp_i - cooldown$

This would make cooldown static which could cripple the algorithm.

$$\lambda = \sqrt[i]{\frac{T_i}{T_0}}$$ ← Need to calculate the starting position of the temperature.

- As temperature T_0 must = starting Temp

- $T_n = 0$

$$\therefore \quad T_i = \frac{T_0}{n}$$ } Any given temperature is equal to 1/n of temp.

$T_0 = 10$

$$T_0 = \frac{\Delta f}{\ln(\frac{1}{p} - 1)}$$

$T_{i+1} = \lambda T_i$

$T_0 = 10$

$T_1 = \lambda 10$

Value unknown

$$\lambda = \sqrt[i=1]{\frac{T_i}{10}}$$

$\lambda = 0$ ... T_0 ← Starting Temperature

$T_i = \lambda^i \cdot T_0$ → Calculate T_0 which is tricky

$$\lambda^i = \frac{T_i}{T_0}$$

once T_0 is calculated you have T_i and λ^i

Initial values are missing and both calculations are recursively connected.

Calculate an average temperature?

Simulated Annealing is reporting worse results than RRHC. The logical step is to take the code apart and investigate all components.

$$\lambda = \sqrt[ITER]{\frac{T_{ITER}}{T_0}} \qquad \lambda = \sqrt[5000]{\frac{0.001}{780.755}}$$

$T_{ITER} = 0.001$ $\qquad \lambda = 0.9973\ldots$

$ITER < 5000$

$T_0 = 780.755$

COOLING RATE CALCULATION: CORRECT

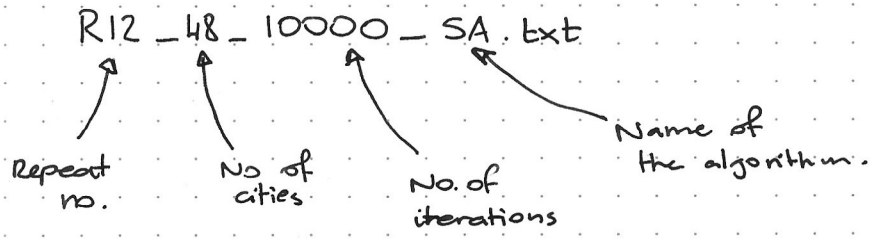
Stephen said that the fitness function could be wrong. Storing the best tour could have been implemented incorrectly.

I am not keeping track of the best solution found!

Major fitness issues were caused by incorrectly referencing the tour - in the end the algorithm was never referencing the tour to be changed. Fixed now.

- Wrote methods to save the results (fitness, mst, qp) for each of the algorithms as comma separated values in a text file. The file is to be used in Excel to plot the results.
- The next stage is to decide how many iterations and repeats each of the experiments is to run for.

- Increase the number of iterations to the maximum of 1 000 000 (1 million).
  → Start at 1000 and go up by 1000 each time.



I hereby consider this project closed.

Johnb Adrian Niemiec

Thu 13th April 2017

- let D be a matrix such that $d_{ij} = d(i,j)$ i.e. the distance between city i and j.
- let the TSP(D) be a solution to the TSP applied to the distances in D
- let the MST(D) be the cost (length) of the MST of D.

→ $f(TSP(D)) \geq MST(D)$

thus, efficiency can be defined as:

$$\frac{MST(D)}{f(TSP(D))} \times 100\% \leq 100\%$$

How do you convert that into code?

→ Use and apply Prims MST from lab 7 to calculate the traversal score of the graph.
→ How to implement the MST and how to use it?

→ Get the MST of the final tour presented by RMHC but how do the two pieces fit together?

↳ Calculate the cost of the tour given the MST?

So given the list of distances convert those into an mst.

→ Leave this for now. Do the algorithms first.

Stochastic Hill Climber

- RMHC can have variable performance, it needs to be improved to escape a local optima.
- To do this we need to let the algorithm accept worse fitness function values during its search.
- The basis of the Stochastic Hill Climbing (SHC):
  - the chance of accepting is a function values during its search
  - a very bad change will have a small chance of being accepted.
  - a slightly bad change will be accepted more often
  - there are many ways of doing this, the example decision function will follow...

- Accepting a new solution according to:

$$Pr(accept) = \frac{1}{1+e^{(f'-f)/T}}$$

f' is the new fitness (line 3 of RMHC)

f is the old fitness

T is the parameter (set to 25 for the 1000 primes scales problem)
→ the correct choice of T can be tricky...

// Stochastic - having a random probability distribution or pattern that may be analysed statistically but may not be predicted precisely.

Random Swap

- Choose two random elements of T where $i \neq j$ and then let $t_i = t_j$ and $t_j = t_i$

RMHC → Fitness drops to 0.0 after ~4k iterations

Why would the fitness drop to 0?

→ The tour gets completely overwritten to the same value. Something is wrong with the swap function.
→ The small change function generates duplicate items → like a virus overwriting things.

With each generation the propagation continues causing more errors.

Fixing the swap function:

a = UI (0 and tour length)

b = UI (0 and tour length)

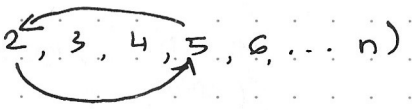
Save a_i as temp.

$a_i = b_j$

b_j = temp

} Flip the positions of those values.

(0, 1, 2, 3, 4, 5, 6, ... n)



Now that the basic algorithm works, it may be time to implement the MST as means of indicating the quality of the final solution. MST is the second variant of assessing the quality.

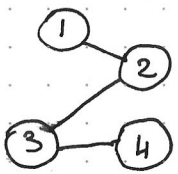
MST:

- a spanning tree that is also a tree (no cycles) that contains all of the nodes of the super-graph. → the edges can only come from the supergraph.
- a graph may have many spanning trees.
- the cost of a spanning tree is the sum of all the edge weights.
- a minimum spanning tree (there may be many) is the spanning tree with the minimum cost.

- the length of the MST can be used as the absolute limit on the fitness of a TSP solution.
- the MST may/will not be a valid TSP solution but the minimum value of a TSP can never be lower that the length of the corresponding MST.

- Distance is being read wrong... wrong co-ordinates are being read.

TOUR



DISTANCE MATRIX

0.0	1.0	2.0	3.0
1.0	0.0	3.0	4.0
2.0	3.0	0.0	5.0
3.0	4.0	5.0	0.0

$d(t_i, t_{i+1})$

- The fitness function is broken due to the way that the code translates mathematical arrays into Java arrays. → We run out of index bounds.

→ Index out of range : 48 to 48

i = 46 ← ?
a = 47
b = 48

Why does i buy out at 46?
Do it in reverse!
Make i count to 48 but correct from the values within the loop.

Tour $\{1, 2, 3, \cdots, n\}$

$d(1,2) == d(0,1)$

i = 0!

Java

↳ Would it be easier to represent a tour from 0 to n?
$\{0, 1, 2, \ldots, n-1\}$

Algorithms and their applications

The travelling salesperson problem

Random Mutation Hill Climbing

Genetic Algorithms and heuristic algorithms

Github

Simulated Annealing and temperature measurement

Graph Traversal Algorithm

How to solve them

MARKING SCHEME

1. DATASETS
The number and size of the datasets are appropriate to support the results.

2. EXPERIMENTAL DESIGN
The number of repeats and number of iterations are appropriate to support the results. They are the same across all implemented methods.

3. EVALUATION
The fitness function and two other techniques are used to judge the quality of the experimental results.

4. METHODS
Four implemented methods work correctly.

5. ACCURACY OF RESULTS
All results are highly accurate (as indicated by the evaluation criteria #3 above)

6. CODE QUALITY (STRUCTURE, COMMENTS, NAMING CONVENTIONS, BUGS)
The code runs and contains no apparent bugs.
The code quality is of an exceptionally high standard.

RMHC

```
for (int i = 1; i <= iter; i++) {
    print(i);

    oldSol = newSol
    oldFit = newFit --> compute new fitness

    make a small change

    compute new fitness

    evaluate the fitness

}
```

return the best fitness.

Algorithm 1. RMHC(iter)
INPUT: iter - ~~the number point in the search space~~ no. of iterations
1) Let S be a random point in the search space,
   let F be its fitness
2) For i=1 to iter
3) Let S' be a random point close to S
4) Let F' be its fitness
5) If F' is better than F then
6) Let S=S' and let F=F'
7) End If
8) End For
OUTPUT: S - a solution

--> What is the fitness calculation which covers all evaluation techniques?

- Need a method which permutes through all the possible tours.
  - → The starting and ending points must be the same.
  - → Check for the difference between standard permutation method and that described by Swift.

Won't the program have to generate one set of permutations at a time?

↳ Or... generate them all and store in an array which is later read one by one.

1. Generate Perms
2. Add start city to end of list
3. Output the list to iterate through

} Iteration process.

→ Is there a simple (recursive?) method to generate all possible permutations?

$PermSize = NoCities!$ $PermLength = NoCities + 1$

Add the start city.

(PROLOG all over again)

Not adding the final city to tour list; this is accounted for in the tour scoring equation.

- How is permutation utilised differently to swap?

Use the length of the minimum spanning tree as an absolute lower limit on the fitness of a TSP solution.

- → The MST may/will not be a valid TSP solution but the minimum value of a TSP solution can never be lower than the length of the corresponding MST.

Let D be a matrix such that $d_{ij} = d(i,j)$

Let TSP(D) be a solution to the TSP applied to the distances in D.

Let MST(D) be the cost (length) of the Minimum Spanning Tree of D

Then we have: $f(TSP(D)) \geq MST(D)$

Efficiency can be defined as:

$$\frac{MST(D)}{f(TSP(D))} \times 100\% \leq 100\%.$$

SOLVING TSP WITH HILL CLIMBING

- Number of cities and the distances between each pair
- A representation and a random starting point
- A fitness function
- A small change operator

? Let P = list of length N, (|P| = N) where $P_i = i$

→ |P| is the length of the tour

→ $P_i = i$ means that P should contain each number of the for loop

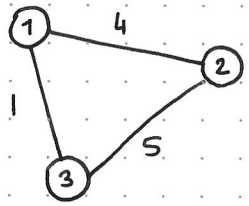
d(i,j)

d(1,2) = 4

d(1,3) = 1

d(2,3) = 5

...



Need a method that will loop through the entire matrix to calculate the tour cost?

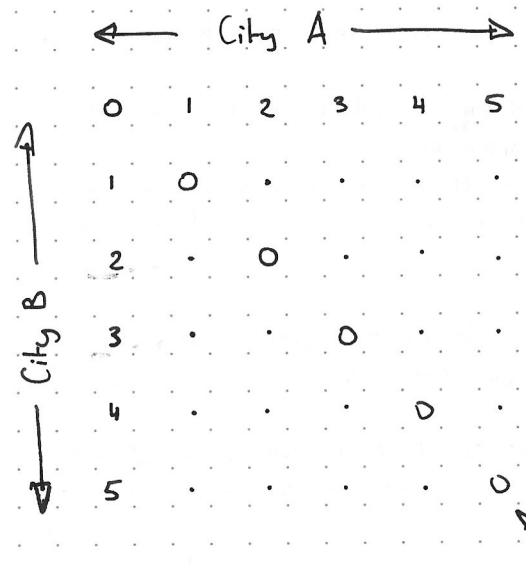
Each TSP_<N>.txt contains the distance between the cities.

↓

Do we need an array to show city numbers?

Each tour is a permutation of integers 1, ..., n

→ Look at the MST!



City distances are stored relative to the matrix.

Given tour coordinates i.e. (1, 2) shouldn't we look at the distance matrix to read the values of that distance.

d(i,j) = d(j,i)

0,1 0,2 0,3 0,4 ... 0,n

1,0 1,1 1,2 1,3 ... 1,n

2,0 2,1 2,2 2,3 ... 2,n

→ How to get a relative distance?

↓

How to map the city matrix onto the distance matrix?

Reverse MST code? ←

Generate a graph of the cities.

→ How to derive city numbers based on the distance scores?

d(A, B) ↓ n = (x, y) ← or y, x? Java is weird!

distance between A & B is the co-ordinates of the distance matrix!

PROBLEM REPRESENTATION

Normally a solution to the TSP problem is represented as a tour in the form of permutation of the integers $1, 2, \ldots, n$.

→ Starting at the left-hand side of the permutation and visiting the cities in the specific order, moving from left to right.
(Ensuring that we do not visit any city twice)

→ A permutation is defined as a shuffling a set of objects.

A, B, C, D → B, C, A, D

Given n objects, we have $n!$ permutations

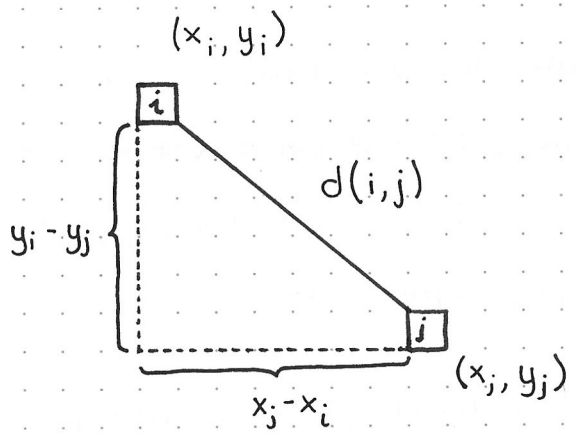
SCORING A TOUR

If we are dealing with n cities then we will define a tour, T, as a permutation of the integers $1, \ldots, n$.

- we will define t_i as the ith value of T (the i^{th} city we visit)
- t_1 is the start and end city.

We will define $f(T)$ as the total cost/distance travelled carrying out that tour.

$$f(T) = \left(\sum_{i=1}^{n-1} d(t_i, t_{i+1}) \right) + d(t_{n*}, t_1)$$



Compute each $d(i,j)$ using Pythagoras' theorem to get the Eucledian distance between city i and j.

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- Each method that is implemented should be run a number of times to remove any one-off results due to the ~~structure~~ stochastic nature of the algorithms.
- Run the experiments for a sensible and logical number of repeats.
- A sample of these datasets should be chosen to represent a well thought out spread of the various sizes of the problems being solved.
- All of the experiments should be run for the same number of iterations, so that they can be compared.
  - → Note that this parameter is up to you but a sensible and logical value should be chosen.

PROVIDING AN INDICATION OF QUALITY OF THE SOLUTION!

→ A suggestion is to use a Minimum Spanning Tree as discussed in the lectures
→ Another is to use the optimal solution files.

TSP - FITNESS FUNCTION

For N cities, let $d(i,j)$ (>0 when $i \neq j$ and $=0$ when $i=j$) be the distance between two cities numbered i and j where $d(i,j) = d(j,i)$ (e.g. the distance between city 1 and 2 is the same as the distance between 2 and 1).

$d(i,i) = 0$ (distance between the city and itself is 0)

$T = \{t_1, t_2, \ldots, t_n\}$ is the tour through the cities as they appear in T, and then we return to t_1.

→ $T = \{3, 1, 2\}$ means that we go through: 3, 1, 2 and then return to 3.

→ No two values in T are the same (can only visit each city once) and that a tour must visit each city. I.e. T.length() == n.size().

Let the length of a tour be defined as $f(T)$ which is the distance travelled when we do the tour.

→ We wish to minimise $f(T)$ - find the shortest tour

$$f(T) = \left(\sum_{i=1}^{N-1} d(t_i, t_{i+1}) \right) + d(t_N, t_1)$$

If $T = \{3,1,2\}$ then:

$$f(T) = \underbrace{d(3,1) + d(1,2)}_{\text{summation part}} + \underbrace{d(2,3)}_{\text{distance from end city to the start city.}}$$

TSP FITNESS FUNCTION (f)

INPUT: N - the number of cities to visit
T - a tour (list of integers of size N)
D - an N x N matrix containing each d(i,j)

1) Let s = 0
2) For i = 1 to (N - 1)
3) Let a = t_i
4) Let b = t_{i+1}
5) Let s = s + d(a, b)
6) End For
7) Let end_city = t_n
8) Let start_city = t_1
9) Let s = s + d(end_city, start_city)

OUTPUT: The tour length s

* Arrays & ArrayLists are zero indexed in Java

The TSP is known as NP-Hard, there is no direct algorithmic or mathematical way to derive a solution in polynomial time. ($O(n^k)$)

We need Heuristic search methods to find a solution:

- Random Mutation Hill Climbing
- Stochastic Hill Climber
- Random Restart Hill Climber
- Simulated Annealing

DEFINING THE PROBLEM

A sales person has to visit 'n' cities.

The aim is to start off at one of the cities, visit each city exactly once and then to arrive back at the starting city. (Tour)

The objective is to find a tour where the sum of the total distance travelled is a minimum.

To solve the TSP we need to know how "far" each city is from each other.

$d(i,j)$ represents the distance from city i to city j.

$$d(i,j) = d(j,i)$$

$$d(i,i) = 0$$

Laboratory 15 - The Travelling Salesman Problem

→ Empirically evaluate how scalable a number of single population heuristic search methods are in solving variably sized problems of TSP.

→ Demonstrate and evaluate how well each of the methods performs as the size of the problem increases.

Objective:

- Produce, present, and report on a Java program capable of solutions to the TSP on a number of different sized problems using different heuristic search algorithms.

- TSP Data zip: Data matrices of size N×N containing distances between the nodes (i,j).

  The distance between (i,j) is the same ~~exac~~ as (j,i).

Tasks:

1) Implement a number of the algorithms to solve the TSP.

2) Compare the algorithms on a number of different sized datasets.

3) Report on the accuracy of the methods as the problem size changes