Traveling Salesman Problem

EC504 Team 1

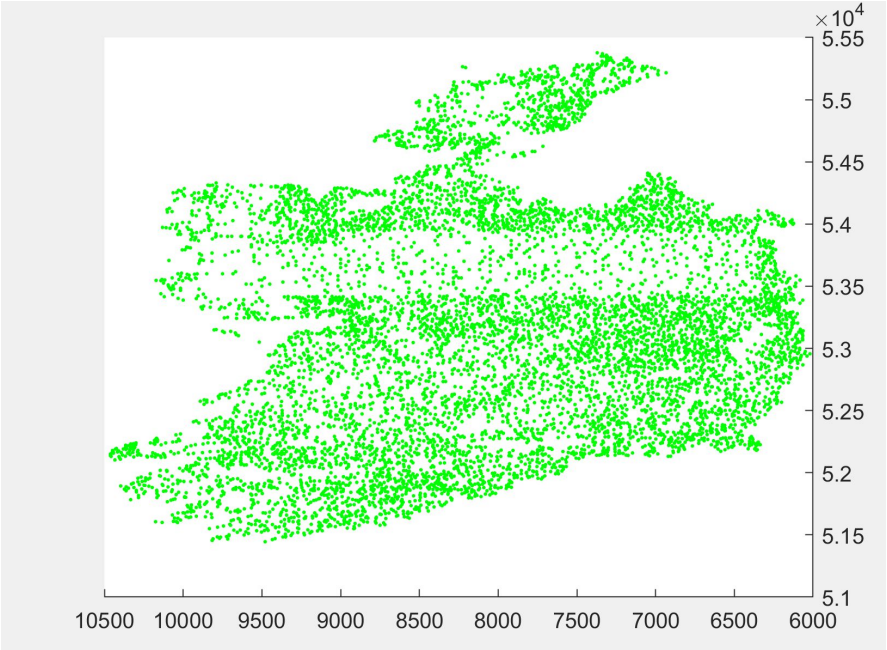
Introduction: Problem Description

Let L = $\sum\sum$ x(i, j) d(i, j); we want to find:

$$L^* = \min \sum\sum x(i, j)\, d(i, j)$$

In other words: Given a **complete, all-to-all connected graph**, find the shortest **Hamiltonian cycle** in the graph.

This is a **NP-complete** problem; no known efficient (i.e. polynomial time) algorithm.



Introduction: Algorithm Preview

Exact:

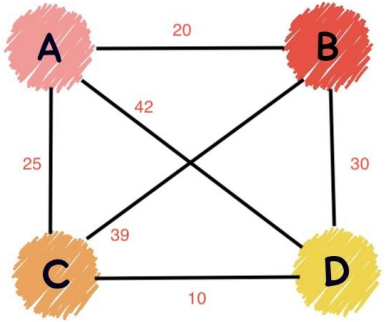
- Brute Force - O(N!); **Impossible for large N**
- Dynamic Programming

Approximate:

- Christofides
- Lin-Kernighan
- KNN
- Simulated Annealing

Brute force Approach

Brute force method: O(N!)

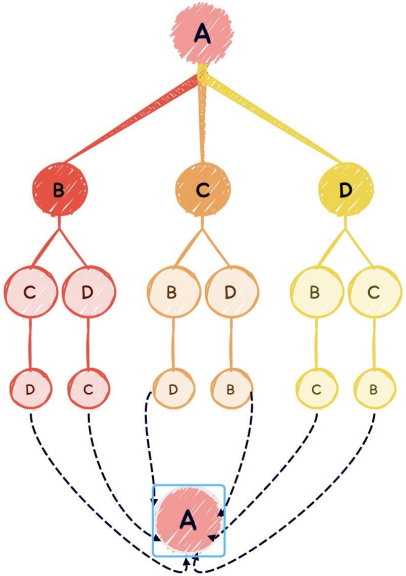


- Divide TSP into subproblems
- Solve the base case: if (all_visited) return dist[pos][0]
- Get the recurrence function:

g(A,{B,C,D}) = Min { dist[A][B] + g(B, {C,D}),

dist[A][C] + g(C, {B,D}),

dist[A][D] + g(D, {B,C}) }



Dynamic Approach

How to avoid overlapping?
→ Use memorization: dp table

dp[2^N][N]: dp[mask][pos]
Min cost of visiting from pos to all other unvisited cities(known from mask) and then to the starting city.

```
tsp(mask, pos)

{

        //Base case: all visited

        If mask=visited_all, return dist[pos][0]

        //Look up dp table first

        If dp[mask][pos] has been calculated, return dp[mask][pos]

        //Try to go to an unvisited city and update dp table

        dp[mask][pos]=min(INT_MAX, dist[pos][city_unvisitied] + tsp(mask_updated, city_unvisited))

}
```

Time complexity:

Subsets number: $2^N * N$
Time for every subset:N

→ $O(2^N * N^2)$

Dynamic Approach

How to represent cities' current state?
→Use bit mask

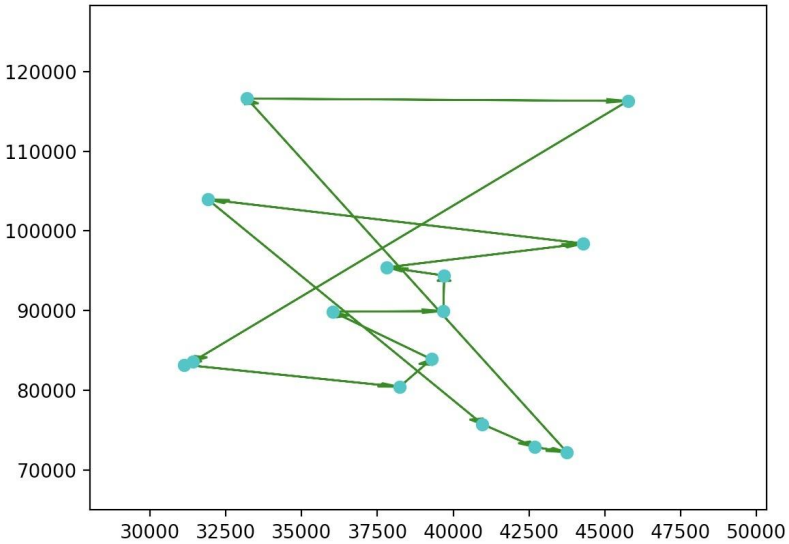
– Concise representation of subsets of small integers {0, 1, 2, 3, . . .}
– e.g: 3 = 0011 in binary represent a set {0, 1} which city 0 and city 1 have been visited
– e.g: 15 = 1111 in binary represent a set {0, 1, 2, 3} which all cities have been visited and it should come back to the start

How to check if the city has/hasn't been visited?
→ Bit operation: Mask & (1<< #city)

–e.g: check if the city 2 has been visited (mask for now is 0110)?
0110 & (1<<2)= 0110 & 0100 = 0**1**00, so city 2 has been visited

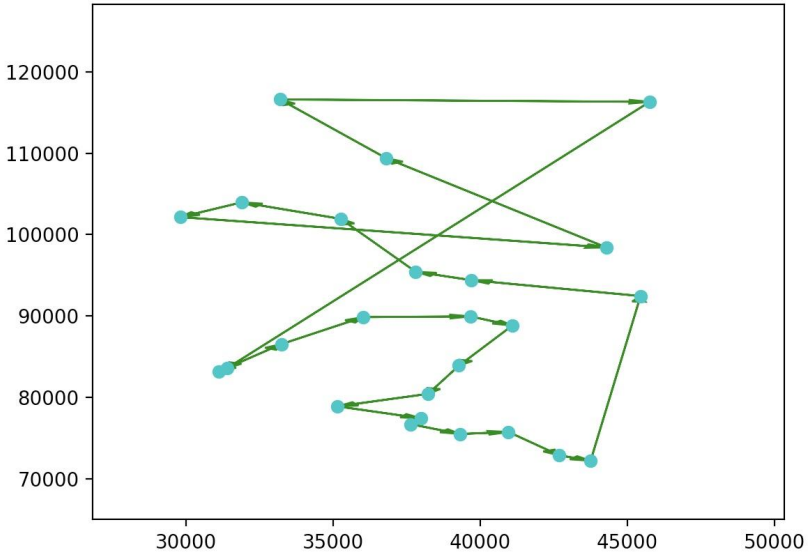
Testing DP method & plotting figure

- Time taken for a dataset with 15 cities: about 0.308914 seconds



Graph for 15 cities

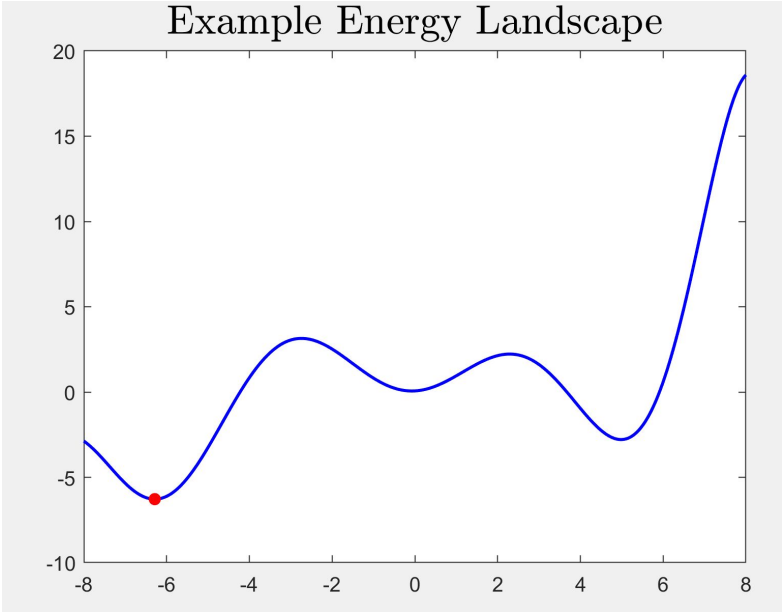
- Time taken for a dataset with 25 cities: about 639.218269 seconds



Graph for 25 cities

Simulated Annealing: Overview

- Cannot exhaustively explore state space - too many combinations.
- ∃ ≥ 1 global minimum, several local minima.
- Let's take an inspiration from physics - **simulated annealing**
  - "Descend" through energy landscape to a minimum energy solution, add randomness to "hop" out of local minima.
- At each time step *t*, propose new configuration S_t, compute energy E_t.
- Lower energy state → *accept configuration*.
- Higher energy state → *accept configuration w/some **probability***.
- **Rationale**: Accepting some higher-energy states may allow us to "bounce" out of local minima.



Simulated Annealing: Implementation Details

(S_0, E_0) = random_path(N)

for t = 1 : STEPS

  while ($n_{attempt}$ < ATTEMPTS && n_{change} < CHANGES):

    $(\Delta E, S_t)$ = update_config(S_{min}, T)

    If $\Delta E < 0$: Accept update $S_{min} = S_t$

    If $\Delta E > 0$: Accept update $S_{min} = S_t$ w/prob. $p = exp(-\Delta E/T)$

    $n_{attempt}$++, n_{change}++ if accept

  end

  $T = T * \alpha$

end

1. update_config - implement update rule for generating new configurations
2. Acceptance probability p
   a. $p \propto \Delta E$
   b. $p \propto T^{-1}$
3. Temperature "cooling" schedule
   a. $T_t = \alpha T_{t-1}$
   b. Exponentially decrease the temperature
   c. Tradeoff between exploration and convergence
   d. Once we are close to a near-optimal solution, we don't want to change the configuration too much

Simulated Annealing: Implementation Details

update_config(S_{min}, T):

Randomly generate two indices $i, j \in N$

Compute change in distance:

$\Delta E = d(i, j) + d(s_i, s_j) - d(i, s_i) - d(j, s_j)$

Reverse path $S_{min}(i+1, j)$** - O(N)

Return ΔE, S_t

Many other such update rules are possible!

Preliminary Parameters:

TMAX = $N^{(1/2)}$

α = 0.95

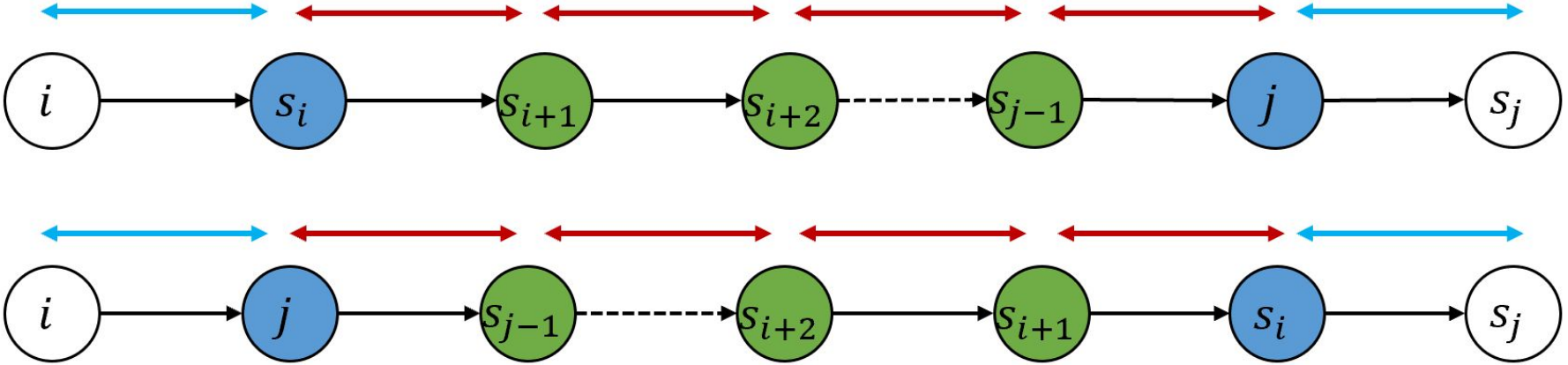
STEPS = log(50 * N)

ATTEMPTS = 100 * N

CHANGES = 10 * N

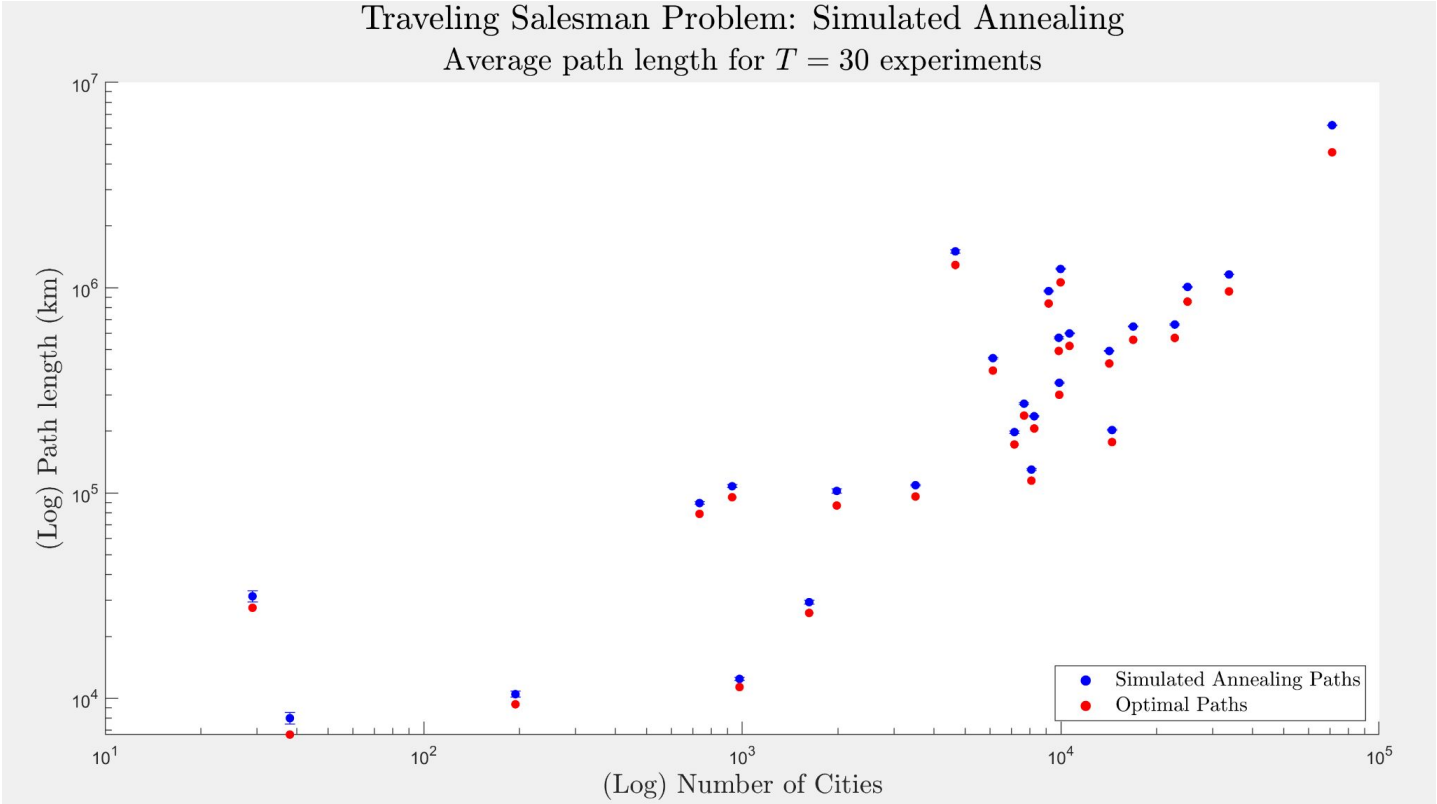
Run time: $T_N = O(N^2 log(N) + N log(N))$

Update Rule: Path Reversal

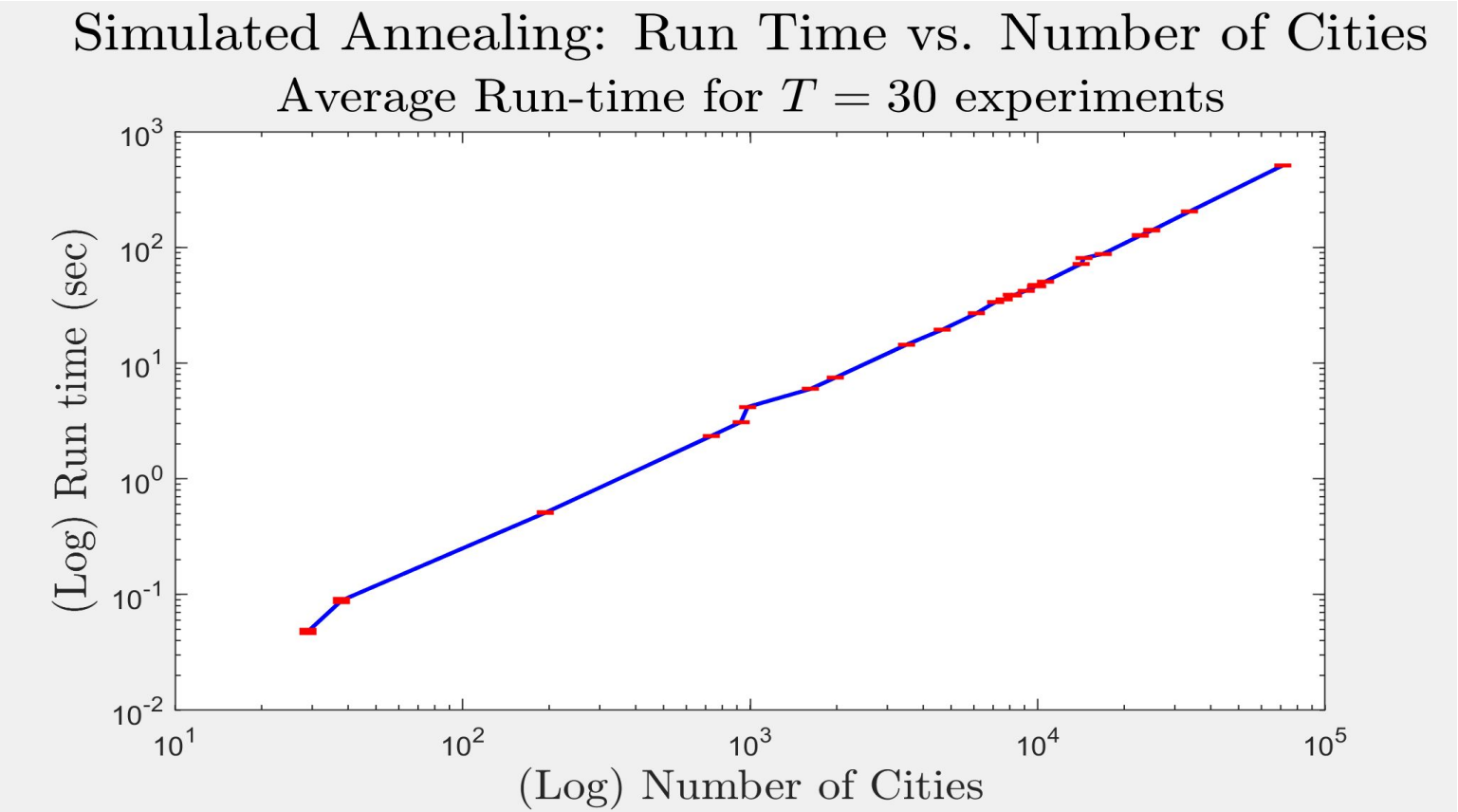


$$\Delta E = d(i,j) + d(s_i, s_j) - d(i, s_i) - d(j, s_j)$$

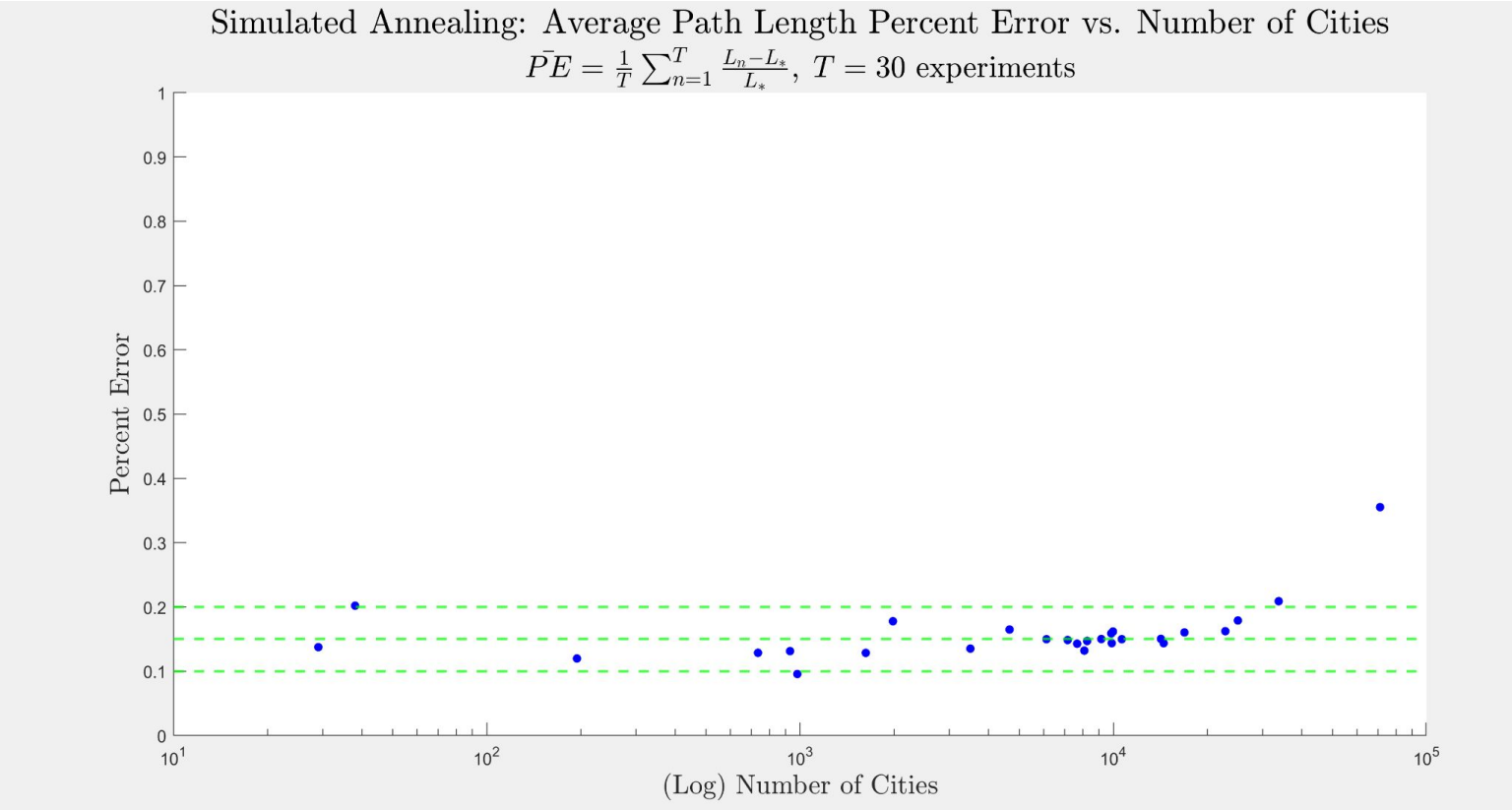
Simulated Annealing: Experimental Results



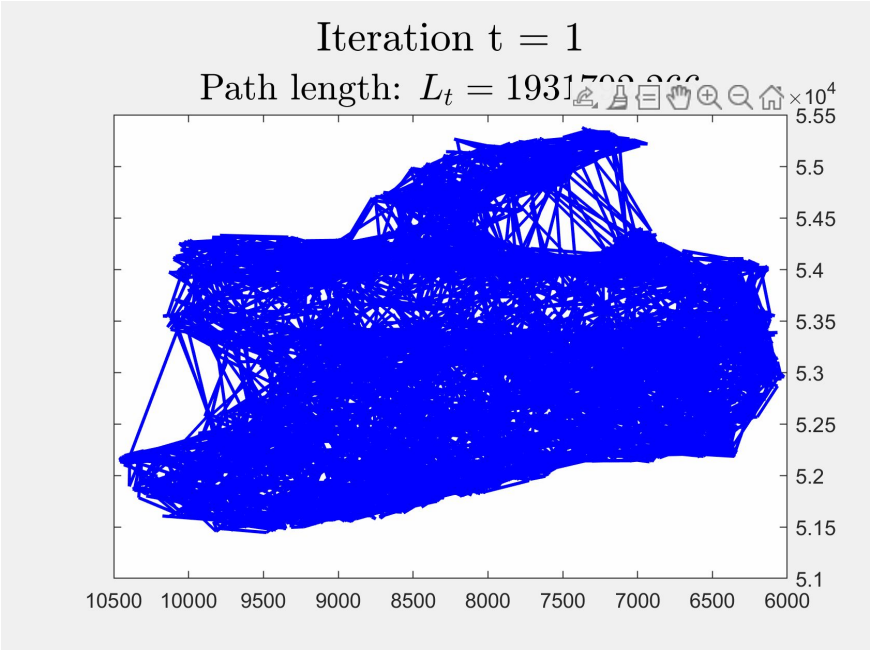
Simulating Annealing: Experimental Results

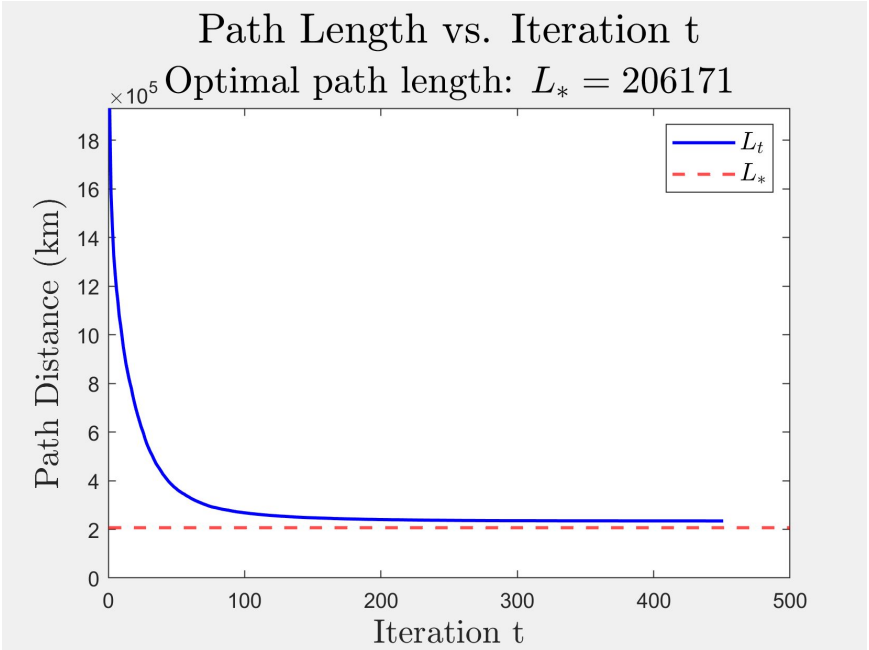


Simulated Annealing: Experimental Results



Simulated Annealing Case Study: Ireland, 8246 cities





Nearest Neighbour algorithm

The nearest neighbour (NN) algorithm is a kind of greedy algorithm. It lets the salesman choose the nearest unvisited city as his next move. This algorithm quickly yields an effectively short route.

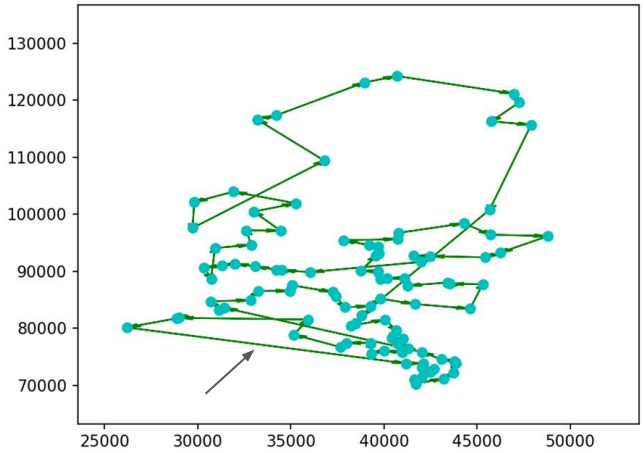
- Initialize all vertices as unvisited.
- Select an arbitrary vertex, set it as the current vertex **u**. Mark **u** as visited.
- Find out the shortest edge connecting the current vertex **u** and an unvisited vertex **v**.
- Set **v** as the current vertex **u**. Mark **v** as visited.
- If all the vertices in the domain are visited, then terminate. Else, go to step 3.

Clearly, the time complexity is O(n^2).

Nearest Neighbour result and analysis

The nearest neighbour algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes which are easily noticed with human insight, due to its "greedy" nature.

As a general guide, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable; if they are much greater, then it is likely that much better tours exist.



Graph for 100 cities

Lin-kernighan core idea:

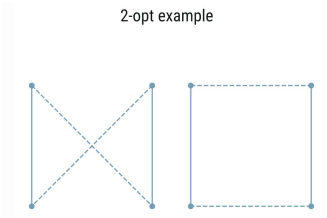
Derive from graph partitioning problem and the basic approach:

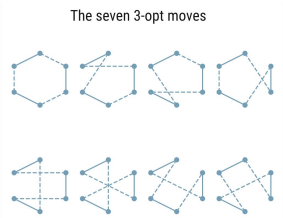
1. Generate a pseudorandom feasible solution, that is, a set T that satisfies it is a tour.

2. Attempt to find an improved feasible solution T' by some transformation of T.

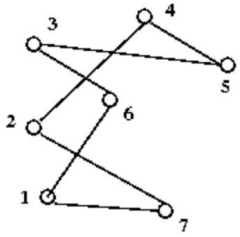
3. If an improved solution is found, i.e., f(T') <f(T), then replace T by T' and repeat from Step 2.

4. If no improved solution can be found, T is a locally optimum solution. Repeat from Step 1 until computation time runs out, or the answers are satisfactory.

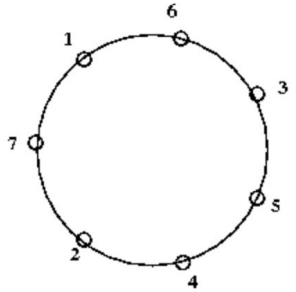




Increase k in sequence and choose the best k

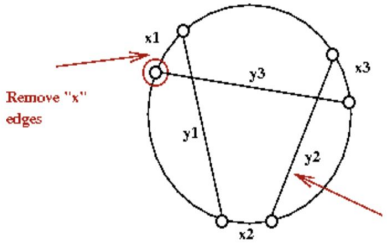


Actual graph and tour

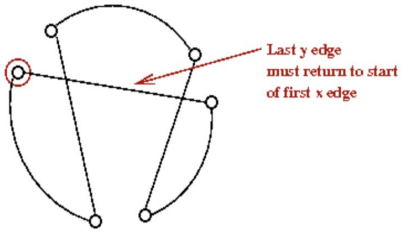


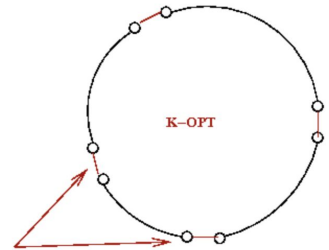
Circular visualization of tour

Current tour

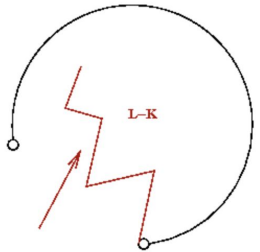


Resulting tour





K–OPT can pick edges anywhere



L–K makes a contiguous path out of "edges in play"

Lin-kernighan details

Detail of step 2:

Suppose that g is the profit or 'gain' associated with the exchange of x and y. Each exchange with x and y, we compute the sum of g=|x|-|y|.

When sum of g is maximum, which means for such starting point, no further improvement can be achieved.

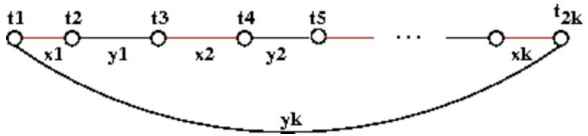
After this, iterate from new starting point until no further reduction.

(*cease at*

$$G = \sum_{1}^{i} g < 0$$

For each iteration:

Lin-kernighan identifies a sequence of edges x1,y1,x2,y2,...yk.



1.Each xi is an edge in current tour

2.Each yi is not an edge in current tour

3.They are all unique

4.Yk are connected to starting point to ensure it's a tour

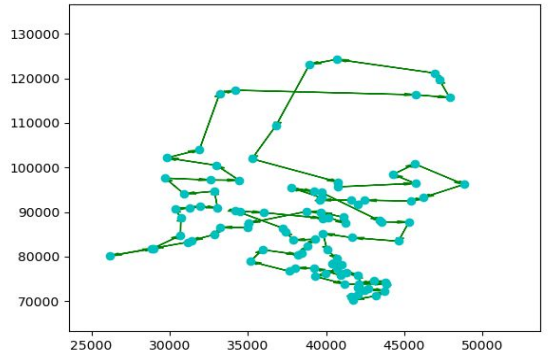
Lin-kernighan results

Test with 100 cities tsp file and use pyplot to generate the path.

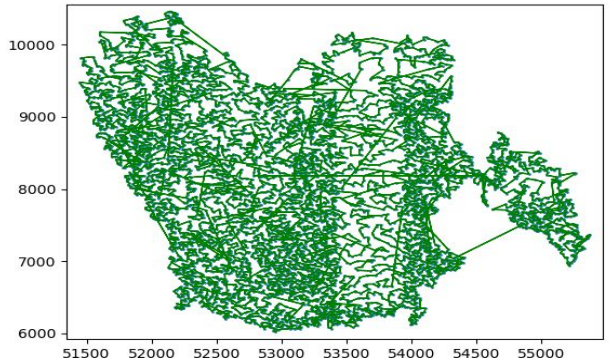
And the time of processing 100 cites using Lin Kernighan is

0.016 seconds.When it comes to 1000 cities,it will cost about 1.6 seconds. The result confirms ***O(n^2.2)***

Generated graph for 100 cities



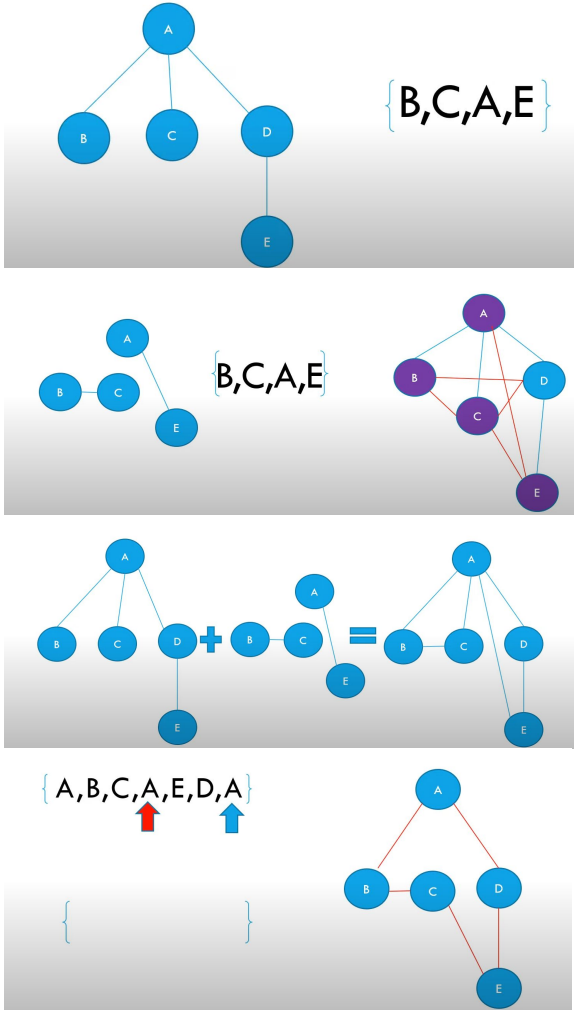
Generated graph for Ireland ,8246 cities



Christofides' Algorithm Basic Idea

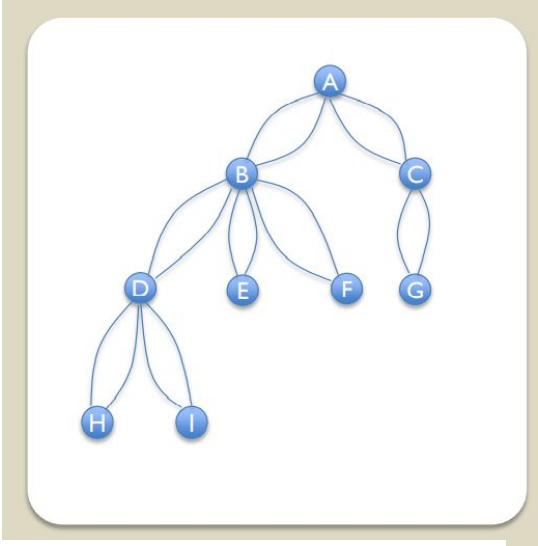
Christofides' algorithm perfect optimization version can guarantee that the final answer will not exceed 3/2 length of optimal path.

1. Make a MST of the graph, prim or Kruskal
2. Find the nodes which have odd edges
3. Make perfect matching of those nodes
4. Find a Eulerian Graph
5. Make sure this is hamiltonian path

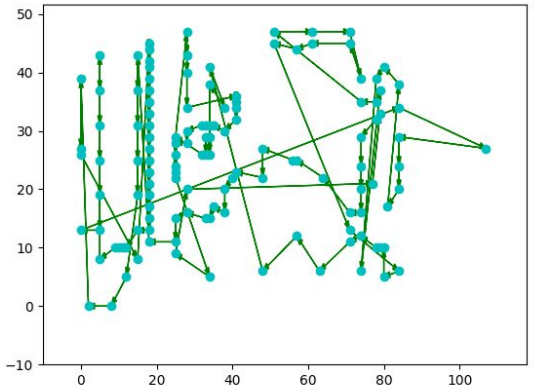


Double-Tree Algorithm Solution

This is the solution to replace point 2 and 3, and easy to travel. I love this idea, however this sucks, without perfect matching, its solution is not comfortable.



For XQF131.tsp, it gave a path that total length is 2e+04, the optimal is 564, it does not fulfil the 3/2 theory Total length of this is 1.1e+03



The complexity of the algorithm is dominated by the complexity of the matching algorithm: O(n^2 * log(n)) if minimal matching is used, or at least O(n^3) or O(nmlog(n)) otherwise. n=V, m=E

Perfect-Matching-like solution

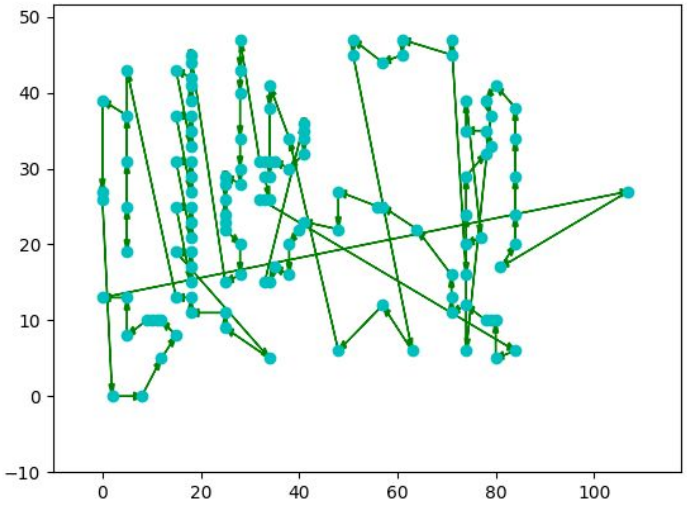
This solution is implement the find_odds and best matching.

Not looking good as the optimal paths

Not yet accomplish the 3/2 theory

the length of the final graph is lower than last one.

Total length:9.8e+02 the optimal path is 564



Next Steps:

- Simulated Annealing:
  - Use stronger parameter set to push estimated path lengths closer to the optimal lengths.
  - Augment the update rule; add local searches/refinements, more random shuffles, etc.
  - Tests with larger number of cities - there is a global dataset!

Lin-kernighan:

Try to do some refinements of Lin-kernighan to make it faster.

Test efficiency on larger dataset.

Christofield:

Make a good perfect matching strategy to improve the result as accurate as possible

Make sure the Christofield’s result has the 3/2 length guarantee.

References

Hansen, Per Brinch, "Simulated Annealing" (1992). Electrical Engineering and Computer Science - Technical Reports. 170.