A Survey of the Traveling Salesman Problem Using Parallel Simulated Annealing and Greedy Heuristics

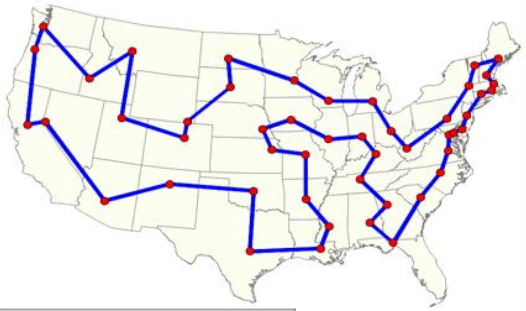
Trevor DeGruccio and Aaron Fox

Part of the Fall 2020 CSE 620 Combinatorial Optimization Class at the University of Louisville
Source Code Repo: https://github.com/aaronfox/Simulated-Annealing-and-Greedy-TSP

Traveling Salesman Problem (TSP)

- The TSP is the face of NP-Complete Hard Problems [1]
- Mathematically Formulated in 1930 by Merrill M. Flood
- The use cases of the TSP stretch far and wide:
    - Finding the shortest path of balancing a network of virtual machine server farms to most efficiently place them for communication
    - Finding an optimal route for a GPS system
    - Modelling which holes to most effectively drill in a circuit board
    - How to efficiently organize book stockers in a library
    - Much, much, more

Algorithms Popularly used for TSP

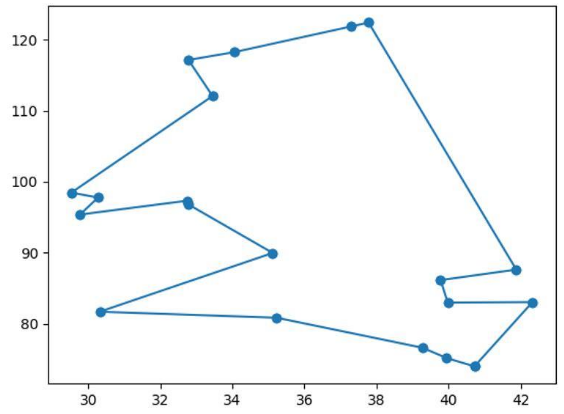


Algorithm	**Time Complexity**
Brute-Force Search	O(n!)
Greedy Algorithm	$O(N^2 \log(N))$
Various Genetic Algorithms	Varies, often $O(N^3)$
2-OPT,3-OPT… K-OPT	$O(N^2, N^3, \ldots N^K)$
Branch and Bound	O(n!)

Traveling Salesman Problem Optimization Explained

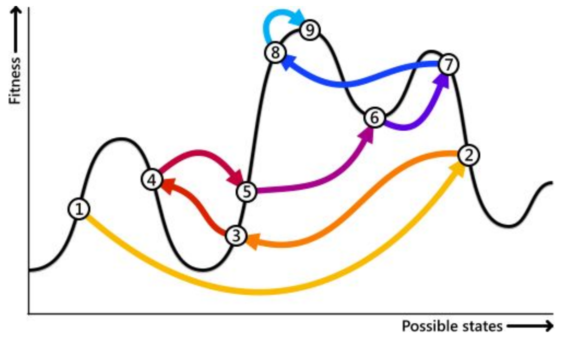
- Goal: find the shortest tour that a salesperson can travel to visit a set of cities exactly once and then return to the starting city.
- Calculating the distance of the tour requires the use of the simple euclidean distance formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



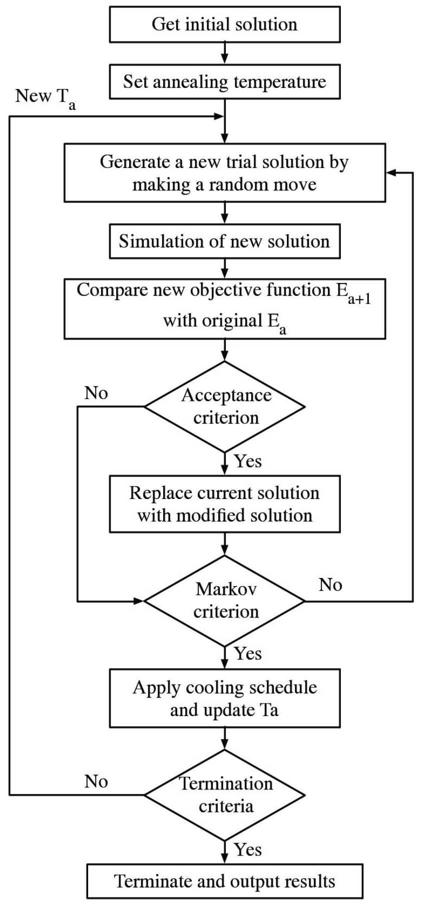
Parallel Simulated Annealing Approach

- Kirkpatrick et al. came up with the idea of simulated annealing in 1983
- Simulated Annealing is a metaheuristic for approximating the global optimum
- The concept mimics the process of misplaced atoms in metal when they're heated and cooled

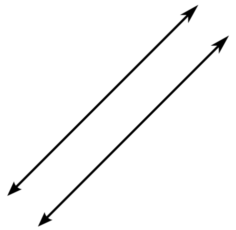


Simulated Annealing Flowchart

This flowchart serves to illustrate the metallurgy metaphor applied to the SA algorithm [2]


Get initial solution
Set annealing temperature
New T_a
Generate a new trial solution by making a random move
Simulation of new solution
Compare new objective function E_{a+1} with original E_a
Acceptance criterion
No
Yes
Replace current solution with modified solution
Markov criterion
No
Yes
Apply cooling schedule and update Ta
Termination criteria
No
Yes
Terminate and output results

Why Parallelize the Code?



- Because of the random nature of Simulated Annealing, sometimes the results are excellent, and sometimes they are not so great.
- Therefore, parallelizing the simulated annealing leads to faster code and allows us to use the most optimal route from all the threads that ran the code,
- More threads running the code = faster and better results
- Made use of the built-in `threading` module in Python

Search Space, Variables, Parameters & Constraints

Search Space for TSP = N! [3]

Variables

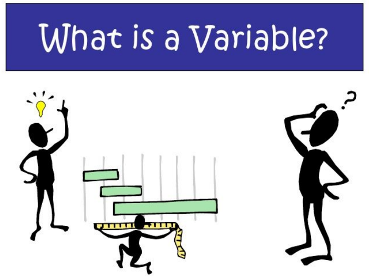
- Initial and Final State, Energy, Temperature, Candidate Generator Function, Acceptance Probability Function

Parameters

- Max Number of iterations, Number of cities, Max and Min Temp, Energy Goal

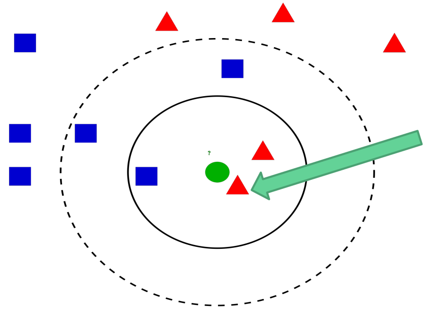
Constraints

- Each city must be visited exactly once and then return to start city



Greedy Approach

- Classical algorithm that is complete and partial in its evaluation [4].
- Also known as the Nearest Neighbor approach
  - Works by starting with a random city, selecting the nearest city, then selecting the nearest city of *that* selected city, and repeating.
  - Neighbor function works by iterating through every city as determined by euclidean. The neighbor that is closest



Starting with the green circle, this red triangle will be selected next as it is the nearest neighbor to the green circle.

Difficulties with Implementation

- Biggest issue with the implementation of both the simulated annealing and greedy approaches was choosing the optimal parameters and variables to use
  - Spent much time tweaking the multivariate equations that arose
- Had to assess the acceptance function of the simulated annealing approach and determine what was optimal
- Had to figure out what the true optimality of the search space was for performance analysis

Optimization Methods

- Much literature about different optimization algorithms used to solve the TSP as described on Slide 3
- In this project, simulated annealing was chosen as the optimization method to compare against the nearest neighbor algorithm, or the local greedy heuristic
- The greedy heuristic connects cities based on the shortest distance from each current city until all cities are connected and a cycle occurs
- This approach is useful to compare against simulated annealing because, while the greedy algorithm may provide a local optima, the simulated annealing approach searches a larger search space although SA takes more time than the greedy approach

Pseudocode For Parallel Simulated Annealing

```
1: For each thread:
2: Let state = initial_state
3: For iteration = 0 to maximum_iteration_num:
4: state_new = neighbor(state)
5: temp = calculate_temp(iteration, temp)
6: If probability of energy of the state and temp
7:             > random variable from 0 to 1:
8:                     state = state_new
9: Return state
```

Pseudocode For Simulated Annealing Continued

Pseudocode for neighbor function:

```
1: Let neighbor be a copy of the current city array
2: Simply swap two cities in neighbor to make it
3: different from original city array
4: Return neighbor
```

Pseudocode for distance used in energy:

```
1: distance = square root of input ((city_2.x -
2: city1.x)
3: ^2 + (city_2.y - city1.y )^2)
```

Pseudocode for candidate generation function:

```
1: If $e^{(\Delta E/T)}$ < random value from (0,1):
2:       Select neighbor candidate determined by
3:       neighbor function
4: Else:
5:       Keep current state
```

Pseudocode for energy of state:

```
1: Energy = 0
2: For i = 0 to length of current city array - 1:
3:       energy += distance(city[i], city[i+1])
4: Return Energy
```

Pseudocode for temperature calculation:

```
1: If T = 0:
2:       Attempt Guess for T based on progression
3:       with steps
4:       Solve for Tmax that give 98% acceptance
5:       Solve for Tmin that gives 0% improvement
```

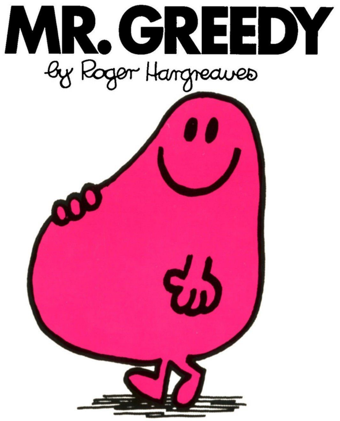
Candidate Generator for Simulated Annealing

- Candidates are able to be pseudo-randomly selected based on probability.
- Effectively, candidate selection is determined by equation

$$e^{\frac{\Delta E}{T}} > \mathrm{rand}(0,1)$$

- At high temperatures, candidate is therefore more likely to be selected even if the change in energy (representing optimality of solution) is not very good.
- This changing at high temperatures allows for escaping local optima

Pseudocode for Greedy [5]



```
1: Do for every starting city and take best route:
2: Initialize array holding indices of cities = indices
3: Initialize array for results including first city = route
4: Mark visited cities array =  visited
5: min = infinity
6: Initialize city indices = city_i, city_j
7: while results array does not have every city and there are still cities left to add:
8:     if path has not be visited and current city < min:
9:             min = dist(city_i, city_j)
10:    increment city_j
11:    Add all paths from city_i to visited
12:    Update final city with best cost visited to route
```

Neighbor Function:

```
1: min_distance = INF
2: min_city = NULL
3: For every unvisited city:
4:     if distance from city to city < min_distance:
5:             min_distance = this distance
6:             min_city = this city
7: return min_city, min_distance
```

Simulated Annealing vs Greedy Algorithm

- Time Complexity wise both the Greedy and Simulated annealing approach to TSP are relatively close

Greedy: $O(N^2 \log(N))$

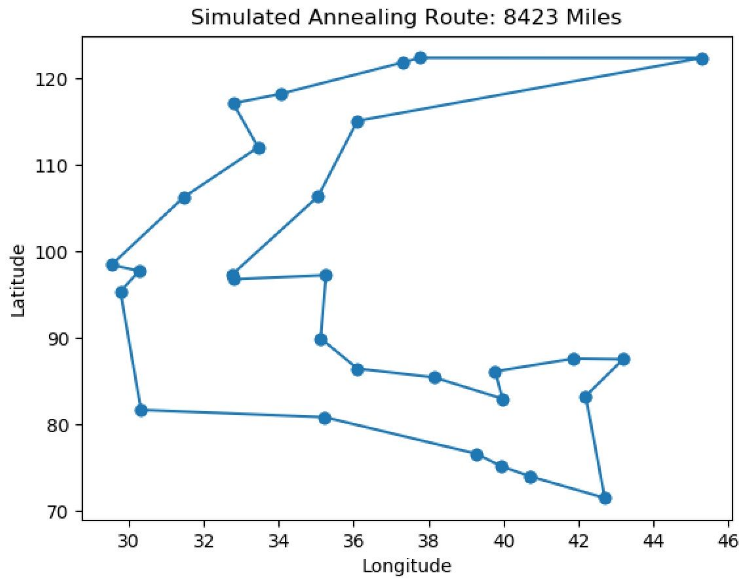
Simulated Annealing: $O((N^2 + N)\log(N))$

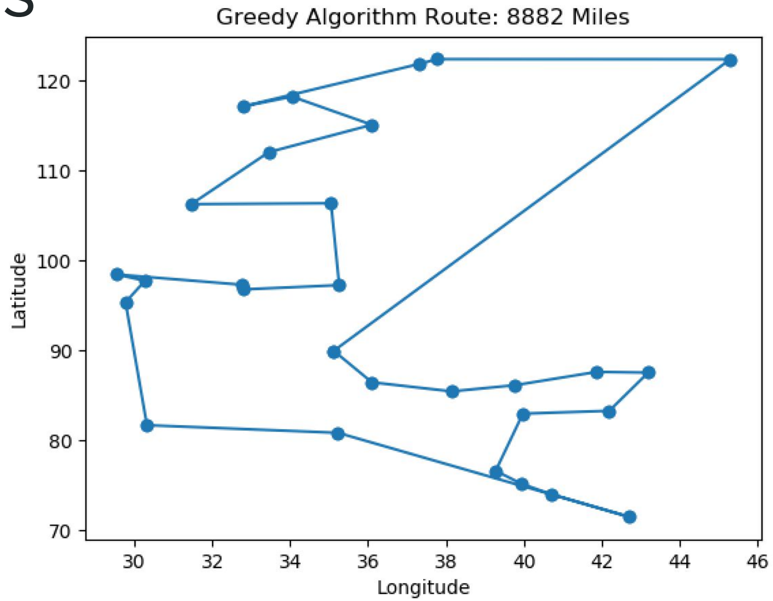
- Simulated annealing also takes $O(N\log(N))$ as shown above, which must be considered as it is determined by the number of steps that occur at low temperatures, which as the graphs in the results section show, consists of most of the steps.

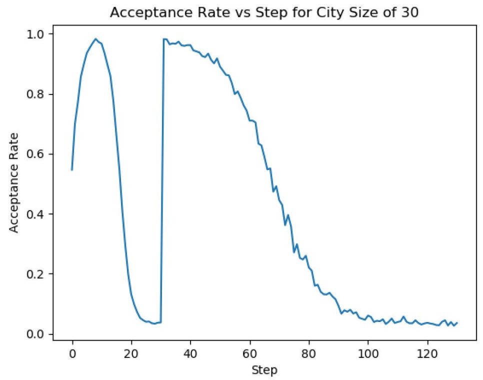
Simulated Complexity Analysis

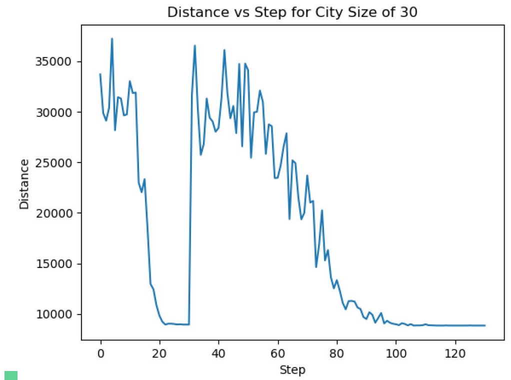
- Using the simulated annealing metaheuristic to solve the TSP requires `O(log(n))` temperature steps, where `n` represents the number of cities
- For each temperature the search examines `O(n)` attempts and accepts any potential changes
- Rejected changes are done in `O(1)` time while accepted changes require `O(n)` reversal in city exchanges
- **Experiment protocol**: To confirm this, the pseudocode was translated to Python along with open-source Python modules to run experiments on these algorithms 10 times each on city sizes of 5, 10, 20, and 30 of the largest US cities; the results were averaged out and tabulated for city sizes as seen on the next slide:

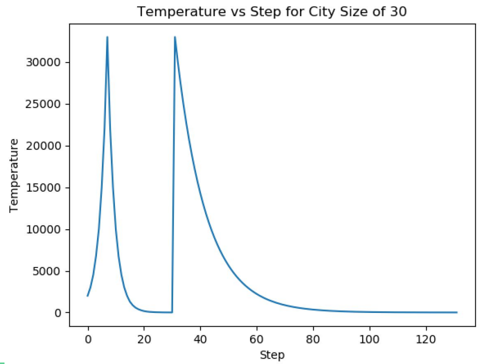
Results for N = 30 Cities











Results Continued...

Number of Cities	**Simulated Annealing (Miles)**	**Greedy (Miles)**	**Difference in Miles (SA - Greedy)**
5	5248	5248	0
10	5898	5924	+26
20	6801	7030	+229
30	8423	8882	+459

Results Discussion

- As more cities are added to the TSP problem set the parallel simulated annealing approach becomes increasingly effective in finding optimal routes compared to the greedy algorithm
- Simulated annealing has a rapid heating and cooling approach that can be seen in the 3 charts at the bottom that allows escaping of the local optima the greedy heuristic gets trapped in, as referenced on Slide 16
- The simulated annealing metaheuristic proves to be more effective and can be applied to many other problem sets that require a global optima to be more closely approximated, making it worth the extra (O(NLOG(N))) time it takes compared to the greedy approach.
- If solely focused on needing an answer very quickly and can accept a less optimal answer, greedy heuristic may be the better choice, however.

References

[1] M. Bellmore and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968. **Slides 2-4**

[2] S. Zhan, J. Lin, Z. Zhang, Y. Zhong, "List-Based Simulated Annealing Algorithm for Traveling Salesman Problem", *Computational Intelligence and Neuroscience, vol.* 2016. **Slides 5-9**

[3] "Is the search space of an optimization problem really that big?," *OptaPlanner*. [Online]. Available: https://www.optaplanner.org/blog/2014/03/27/IsTheSearchSpaceOfAnOptimizationProblemReallyThatBig.html#:~:text=Traveling Salesman Problem (TSP): The base search space is,of the number of lectures. https://www.hindawi.com/journals/cin/2016/1712630/. **Slides 8**

[4] G. Gutin, A. Yeo, and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP," Mar. 2002. **Slides 8,12**

[5] OpenDSA Project, "Reduction of Hamiltonian Cycle to Traveling Salesman¶," in *OpenDSA Data Structures and Algorithms Modules Collection*, OpenDSA Project, Ed. . **Slide 14**