Brice Wilbanks
COMP361 Assignment 4
October 2, 2019

Linear Programming Simplex

Linear Programming → Simplex Assignment 4

Max $Z = 2x_1 - x_2 + x_3$

s.t. $3x_1 + x_2 + x_3 \leq 60$

$2x_1 + x_2 + 2x_3 \leq 20$

$2x_1 + 2x_2 + x_3 \leq 20$

$x_1, x_2, x_3 \geq 0$

Slacks

$-2x_1 + x_2 - x_3 + Z = 0$

$3x_1 + x_2 + x_3 + s_1 = 60$

$2x_1 + x_2 + 2x_3 + s_2 = 20$

$2x_1 + 2x_2 + x_3 + s_3 = 20$

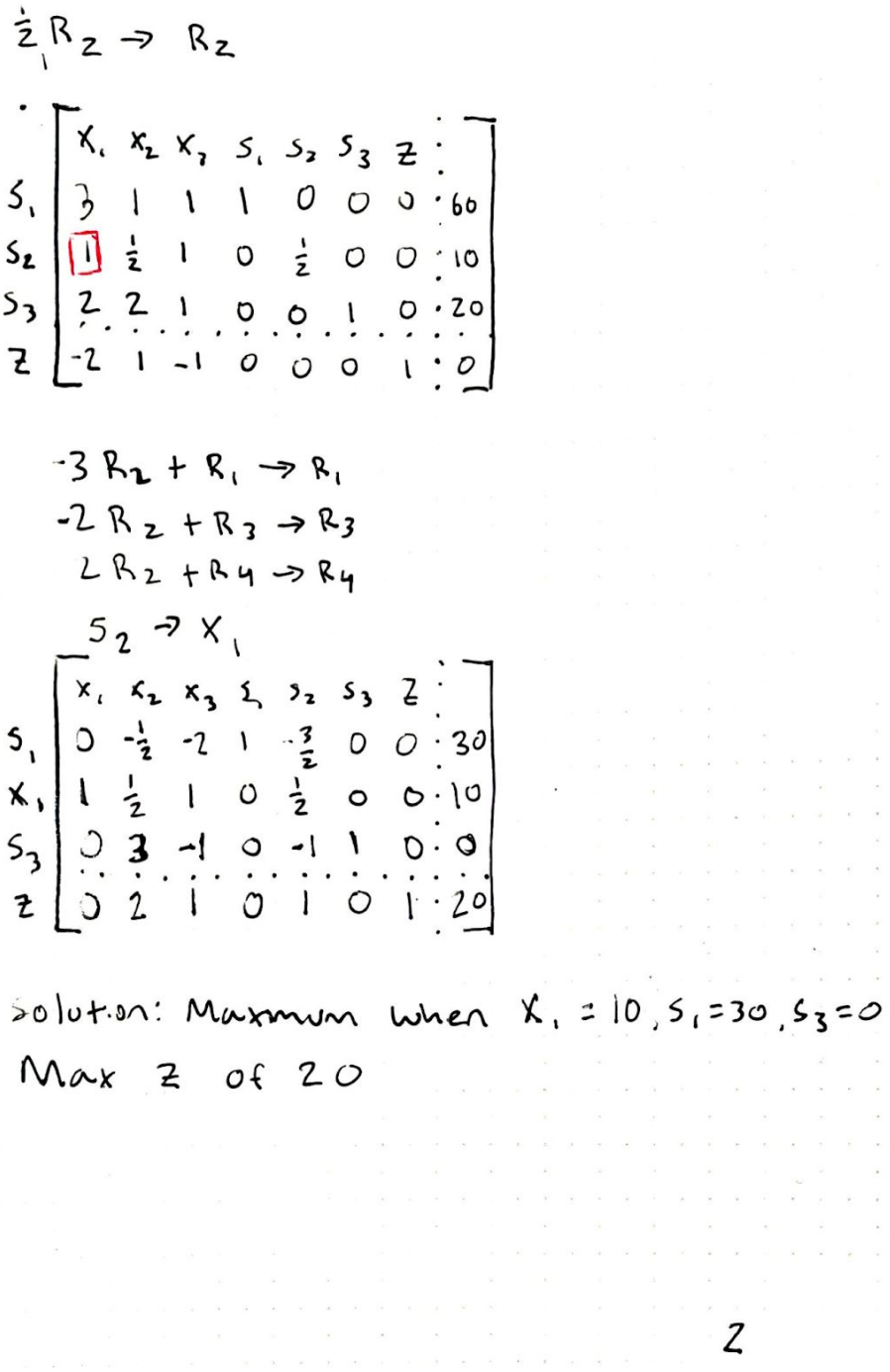
Table

	x_1	x_2	x_3	s_1	s_2	s_3	Z		Pivot
s_1	3	1	1	1	0	0	0	60	$\frac{60}{3} = 20$
s_2	2	1	2	0	1	0	0	20	$\frac{20}{2} = 10$
s_3	2	2	1	0	0	1	0	20	$\frac{20}{2} = 10$
Z	-2	1	-1	0	0	0	1	0	

Pivot Col

Pivot: Row 2, col 1

$\frac{1}{2} R_2 \rightarrow R_2$



Breaking the Simplex Algorithm

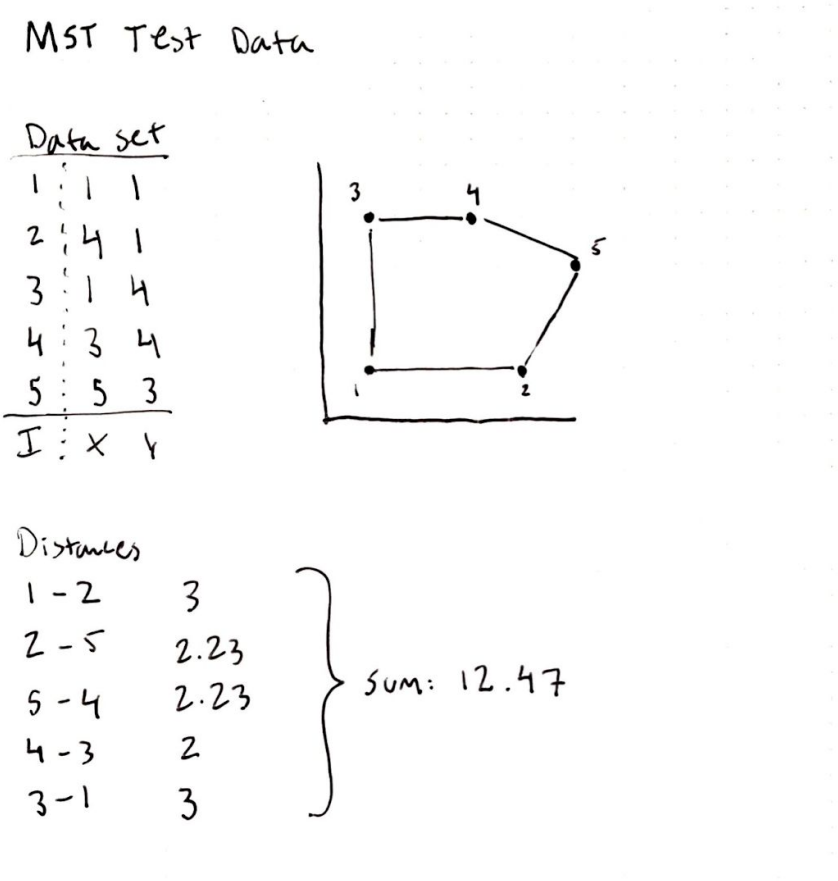
The best way to break the simplex algorithm is to flip the inequalities, changing the problem to saying you must have at least some constant k of some item x. This is opposed to saying you must less than some k constant of x item. When this occurs, we can continually count up and add items, but we can never go below the minimum requirement. Therefore, the problem becomes unbounded and unsolvable via the simplex algorithm.

MST Heuristic

To implement the approximation algorithm for the Travelling Salesman Problem, Prim's MST algorithm was used to create the shortest path. Prim's is a greedy approach to the MST problem and will choose the shortest path to the next unvisited node at every given node. Therefore, this algorithm is not optimal by definition.

Testing

To ensure the algorithm was working as expected, the following test data was used.



The results of this data can be seen under `/mst-heuristic/output/test.txt` as well as below:

```
## TSP OUTPUT FILE - APPROXIMATION ALGORITHM ##
NAME: TEST
Start City: North Decatur

North Decatur --> New Rochelle         Distance of:3.0
New Rochelle --> Pace         Distance of:2.23606797749979
Pace --> Cedar Falls         Distance of:2.23606797749979
Cedar Falls --> Los Gatos         Distance of:2.0
Los Gatos --> North Decatur         Distance of:3.0

End of Tour! Total Distance of: 12.47213595499958
```

Tests on TSP Libraries

To test the algorithm, three TSP Problems were used from the dataset. Below is a table showing the dataset name, the optimal solution, and this MST algorithms solution. The last column shows what percentage the MST Heuristic is of the optimal solution. Essentially, anything over 100% is not optimal and anything under 100% would be better than the theoretical best.

Dataset	Optimal	MST Heuristic	% of Optimal
`a280`	`2579`	`3148.109934934404`	`122%`
`berlin52`	`7542`	`8980.918279329191`	`119%`
`kroB200`	`29437`	`36981.59014085329`	`125%`

Running the Program

This section of the program and the results shown above are found under the `/mst-heuristic/output` path. To run with a new dataset, paste the full path in the method call on `line 131` of the `mst-heuristic.py` file.

Simulated Annealing

To implement the simulated annealing solving version of the travelling salesman problem, a few notes were made in the development of the algorithm.

Initialisation Method

To initialize the algorithm, the program firsts creates the initial tree. The initial path is in no particular order, but starts from the first index node listed in the TSP datafile. The initial path just follows this file provided, regardless of path distances. Once this is complete, the tours initial distance is calculated and stored before beginning the algorithm.

Cooling Schedule / Stopping Schedule

The algorithm follows the traditional temperature based schedule for finding a close to optimal solution. The initial temperature starts at 20 before cooling to 0. The stopping criteria are defined when the temperature reaches 0. This indicates that we are as close as possible to an optimal solution. The temperature decreases by a function of the iteration each loop. Temperature is defined as the following:

$$temp = temp - \frac{1}{log(iteration+2)}$$

The cooling schedule decreases logarithmically and as a function of iteration. With this, it decreases fast at first and slows down as the iteration increases. The temperature is also used in the generation of the path rejection probability. Under this method, we will accept the path with the following probability:

$$probability = e^{(L1 - L2)/temperature}$$

where L1 = current calculated distance

L2 = new calculated distance

Under this method, L1-L2 will always evaluate to a negative value since the new distance will always be longer in order to reach this gate.

Neighbourhood Definition

To generate a new path, the neighborhood is defined simply as two nodes in the city sequence. With this simple definition, we can choose two cities to be swapped in the tour, changing the total path length. The logic lies in that we make one small change at a time until we reach our optimal, not huge, sweeping changes.

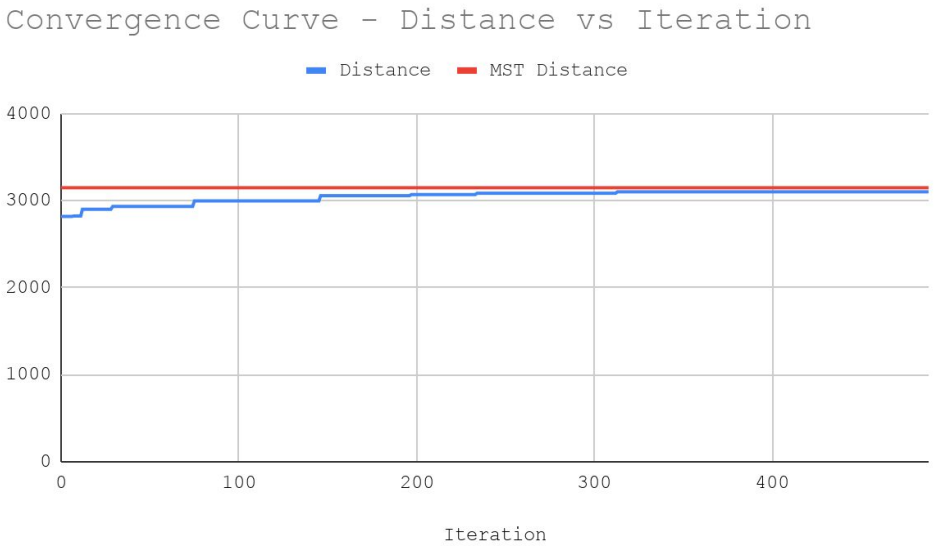
Tests on TSP Libraries

To test the algorithm, three TSP Problems were used from the dataset. The same datasets used above were used on this problem to compare the effectiveness of the algorithms. Below is a table showing the dataset name, the optimal solution, the above MST algorithms solution, and the local search algorithm solution. The last column shows what percentage the simulated annealing is of the optimal solution. Essentially, anything over 100% is not optimal and anything under 100% would be better than the theoretical best.

Dataset	Optimal	MST Heuristic	Simulated Annealing	% of Optimal	% of MST
a280	2579	3148.1099	2820.4327	109%	89.5%
berlin52	7542	8980.9182	22270.0594	295.28%	247.97%
kroB200	29437	36981.5901	327452.3654	1112.3%	885.44%

Convergence Curve

The results of the simulated annealing algorithm can be best seen with a convergence curve. Below is a chart comparing the iteration to the distance achieved by the SA algorithm.



While the expected result may be a generally decreasing curve, the results of the SA (shown in blue) trend toward increasing. The reason for this is the algorithm getting trapped in a local maximum. As explained under the genetic algorithm analysis as well, the direct index tour of 1→280 is a near optimal solution with a distance of 2818.62, which is only 9% worse than optimal. Since this is a local minimum solution, it makes the chances of jumping over a local maximum very small. The a280 dataset is therefore not the best dataset to show the SA algorithm works as expected. However, this local minimum of 2818.62 is slightly better than the greedy MST solution (shown in red).

Running the Program

This section of the program and the results shown above are found under the `/simulated-annealing/output` path. To run with a new dataset, paste the full path in the method call on `line 184` of the `simulated-annealing.py` file.

Genetic Solving Algorithm

The genetic algorithm was solved using the crossover and swap mutation methods described in the course slides.

Data Structures

To store the tours generated later in the problem, the program uses the Python list data structure for each tour. Each node within the list is a Python dictionary containing the city name, the original index ID, and the X/Y point provided by the dataset. This node is copied, not referenced, in each generated tour.

Initialization / Generation

To generate the initial population, we first generate the basic, `1→N` list where each node goes to the next indexed node. After this is generated, we then `n/10` generate random tours to begin the population. This provides enough tours to begin the genetic algorithm.

Evolution Process / Selection Operator

During each iteration of the algorithm, the shortest distance of the population is chosen as the parent. This parent is then used in the crossover and mutation algorithms to create children to the parent.

Crossover / Process and Rate

The selected parent tour is put into the crossover algorithm `n` times to generate n/10 children. Under this crossover procedure, a section of the tour from a randomly selected, noted `i → k` of some `n` length. This chosen section is then copied to the new child. There is only one child created for each `i → k` selection. The new child is then padded with a random order of the remaining, not previously included, nodes. This process creates a random child tour that all contains the sequence provided from the parent. Crossover occurs at every iteration and creates `n/10` children nodes

Mutation / Process and Rate

After the crossover creates a new child tour, there is a 10% chance that it is mutated. This algorithm uses swap mutation, where two random nodes of index 0 to n are chosen and swapped. This is similar to the simulated annealing algorithm implemented above. The chance of mutation occurs at every iteration.

Stopping Criteria

To control the iterations, the same stopping and cooling schedule used in simulated annealing was used. The temperature starts at a predetermined constant T and decreases at a logarithmic rate until zero. In some samples, it was more effective to use a linear decrease function, where the temperature decreased by half a step each iteration. In the logarithmic decrease rate, it is a function of the current iteration plus some constant. Once the temperature is zero, the algorithm stops running.

Tests on TSP Libraries

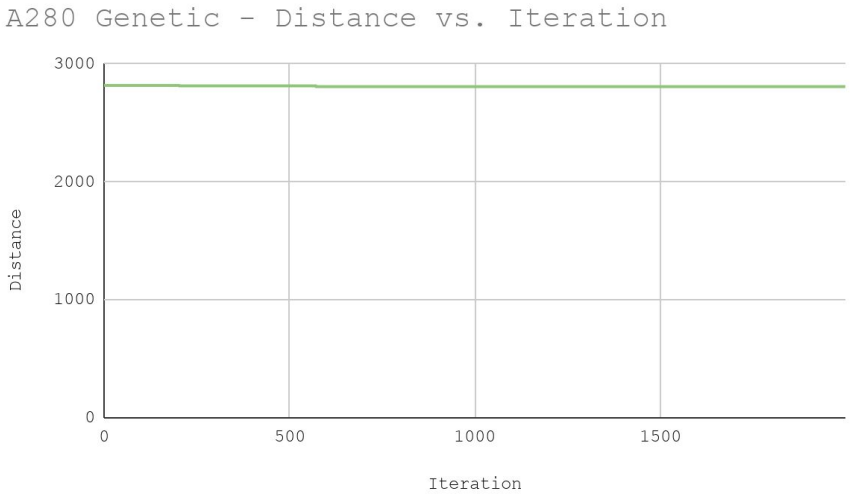
Dataset	Optimal	MST Heuristic	Simulated Annealing	Genetic	% of Optimal
a280	2579	3148.1099	2820.4327	2812.2819	109%
berlin52	7542	8980.9182	22270.0594	11831.9723	156.8%
kroB200	29437	36981.5901	327452.3654	250878.2536	852.25%

The A280 dataset performs about the same as the simulated annealing algorithm as it does for the genetic algorithm with the genetic path being less than 0.2% shorter from the SA algorithm.

The Berlin52 dataset performs well with the genetic algorithm compared to the SA algorithm. The improvement is a path almost 50% shorter than the SA algorithm. However, genetic does not perform as well as the MST Heuristic algorithm for this dataset.

Convergence Curve

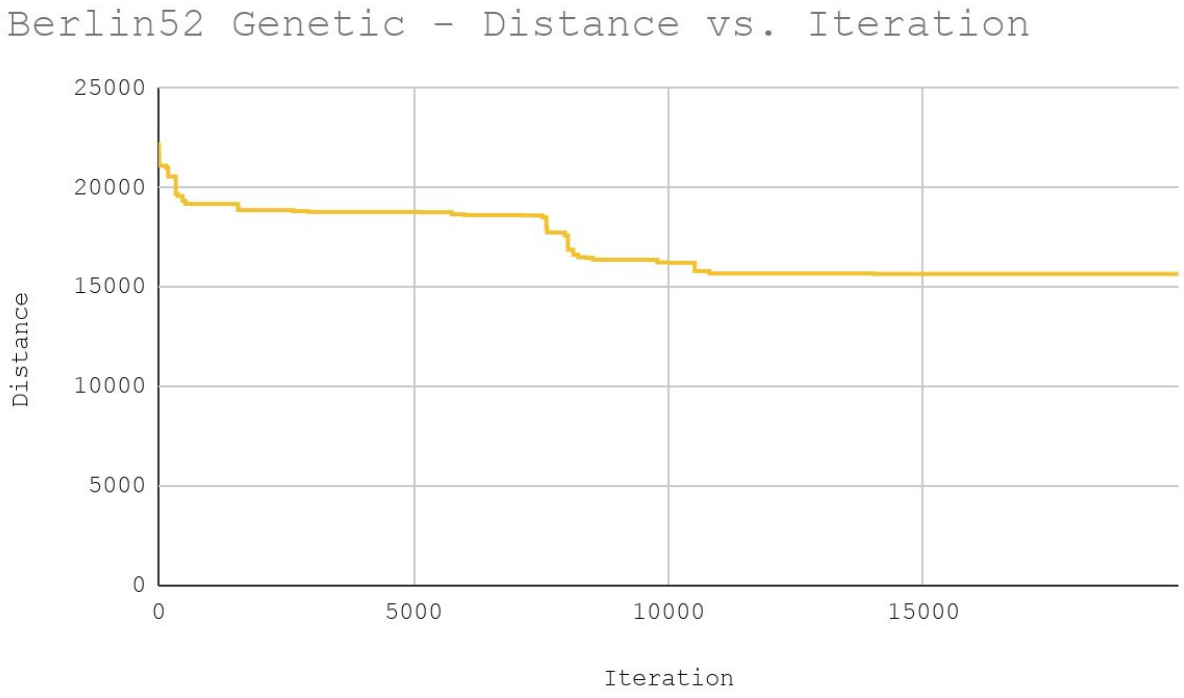
Below is the convergence curve for the a280.tsp dataset. Interestingly, the a280 dataset is almost optimal with a path that is close to 1→280 without any variation. Because of this, a better, more optimal solution is incredibly hard to find since the first initial tour is often the best. The MST local search algorithm does not find this result since it is greedy my nature and does not compile an initial tree from 1→280. For this dataset, the genetic algorithm performs better than the MST algorithm.



The algorithm only finds four solutions through its 2000 iterations. Those are listed in the table below. As explained above, the first solution of 1→280 is a near optimal solution and is therefore kept as the elite parent for a long time. Crossovers and mutations only produce slightly better solutions.

2818.621642	2814.869428	2814.800495	2808.472862

To show the algorithm works as expected, below is the convergence curve for the berlin52.tsp dataset. This dataset is not optimal with a 1→52 direct tour, therefore an optimal solution is found genetically and is seen with a distance vs iteration curve. The best result found for this iteration was 15671.6 units with an initial temperature of 10000 and a linear step decrease of 0.5 each iteration., creating a total of 20000 iterations.



Running the Program

This section of the program and the results shown above are found under the `/genetic/output` path. To run with a new dataset, paste the full path in the method call on `line 254` of the `genetic.py` file.

Hamiltonian/TSP Problem Complexity

Assume the optimal solution to a symmetric TSP (sTSP) instance has length L. Given a walk, W, show that it is possible to check if W is an optimal solution to this symmetric TSP instance in polynomial time. This proves that sTSP ∈ NP. (5%)

Simply put, we can assume that the optimal L for sTSP is given to us. Therefore with this L, we can traverse the n elements of the TSP tour to see if the W is > or == the optimal L. If it is ==, we have

proven that W is an optimal solution in polynomial time since there are n elements. If W is > L, we have proven that W is not an optimal solution and therefore sTSP is not $\in$ NP.

Show that the Hamiltonian cycle problem (HAM) is polynomial-reducible to the symmetric travelling salesman problem. That is, prove HAM $\leq_P$ sTSP. (10%)
First, we must show how to convert instances of the Hamiltonian Cycle (HAM) into instances of the Travelling Salesman Problem (sTSP). We can prove this with the givens: since HAM is a cycle which visits every node in the cycle exactly once; and a TSP is the same, we prove that a HAM is the same definition as a sTSP.

Second, we can convert solutions for sTSP into solutions for HAM, and prove that those conversions can be done in polynomial time. We have already proved that sTSP and HAM are one of the same and we have proved that the optimal solution can be verified in polynomial time, due to there being n elements. Therefore, we can conclude HAM $\in$ P also.

Given HAM $\leq_P$ sTSP and sTSP $\in$ NP, what can we thus conclude about the complexity class of sTSP?
As we proved above, HAM $\leq_P$ sTSP (that is, HAM is polynomially reducible to the sTSP). Knowing that sTSP is an element of NP problems and that HAM is an element of NP-Hard problems. Therefore, we can fully concolute that the complexity class of the sTSP problem is in-fact NP-Complete. By definition, since every problem of HAM can be reduced to sTSP, the problem is NP-Complete.