1 INTRODUCTION

The Traveling Salesman Problem (TSP) is an NP-Complete computational problem which has been studied extensively for several decades [2]. In this paper, techniques for obtaining/approximating the solution to the symmetric Traveling Salesman Problem are developed and compared. A branch-and-bound algorithm using a minimum spanning tree to inform a lower bound on sub-problems is presented; which, given enough time, produces an exact solution to the problem. The use of approximation algorithms proves more prudent for larger problems while sacrificing solution quality; a fact which motivated the development of a greedy local search algorithm which explores 1-exchange neighbors. In addition, two local search algorithms a 2-Opt Hill Climb and Simulated Annealing Algorithm are explored. These local search algorithms are ideal for finding low error solutions in limited time, although no performance bounds are guaranteed beyond what is expected from the Greedy Algorithms that initially seed them. The results derived showed that the local search algorithms were extremely adept at providing relatively good solutions even in the presence of large data sets, however they struggled to consistently return optimal solutions even for the smaller data-sets. Of the two local search algorithms explored, the simulated annealing algorithm provided clear benefits especially for larger data sets which contain a higher number of local optima which can frequently trap other algorithms in a non optimal neighborhood. *Note: For the purposes of this project the Simulated Annealing and Approximation Algorithms are considered to be the two primary algorithms for scoring purposes, with the Branch-and-Bound and 2-OPT Hill Climb serving as the extra credit algorithms.*

2 PROBLEM DEFINITION

The symmetric traveling salesman optimization problem is formalized as follows:

Given a connected undirected graph G consisting of n nodes, $\{v_0, \ldots, v_{n-1}\}$, with edge weights d_{ij} between nodes i and j. Find a Hamiltonian Cycle, a path P* where each node has degree 2, with minimum weight.

In this paper, nodes in a 2 dimensional space were considered: $v_i = \begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix} \in \mathbb{R}^2 \quad \forall i \in [0, n-1]$

With edge weights calculated using the Euclidean distance: $e_{ij} = ||v_i, v_j||_2 = \sqrt{(v_{ix} - v_{jx})^2 + ((v_{iy} - v_{jy}^2))} \quad \forall i \neq j, i \in [0, n-1], j \in [0, n-1]$

The solution is an element in the set of vertex sequences which are Hamiltonian Cycles given by:

$\mathcal{H} = \{(v_i)_{i=0}^{n-1} : v_0 = v_{n-1}, v_i \neq v_j \quad \forall i, j \in [0, n-1]\}$

The symmetric TSP is given by following optimization problem:

$P^* = (v_i^*)_{i=0}^{n-1} = \underset{(v_i)_{i=0}^{n-1} \in \mathcal{H}}{argmin} \quad \sum_{i=0}^{n-2} ||v_i, v_{i+1}|| + ||v_0, v_{n-1}||$

3 RELATED WORK

The Traveling Salesman Problem has been a widely studied in the fields of graph theory and computational algorithms for almost 200 years. Long considered impossible, formal methods for restricting the problem space using approximation algorithms, similar to the 2MST approximation used here. With formal methods to approximate routes to given accuracies solving the problem for large data-sets became feasible in practice. With a good starting point for lower bounds, methods such as the Branch-and-Bound method used in this paper could now solve large data-sets by limiting the search space using the previously calculated lower bounds given by the approximation guarantee. Recently in 2011 researchers from Stanford and McGill Universities were able to improve upon the approximation algorithm developed by Nicos Christofides in 1976 which had defined performance bounds of at most 50% greater than the true optimal solution when they developed an approximation technique which improved upon Christofides' algorithm by four hundredths of a trillionth of a trillionth of a trillionth of a trillionth of a percent, a staggeringly small value in most other scenarios. Although this improvement was relatively minimal, it broke open a wall that existed in the world of computer science for over 35 years, proving that in practice a better solution was possible. With the rise of Quantum Computing improved results for the Traveling Salesman Problem will not only be made possible by improved algorithms, but also by the exponential increase in processing power that the Quantum Technology brings. With this technology on the rise we are poised to see major improvements in the runtime of NP-Hard algorithms such as the TSP problem which will be extremely important given the increase in the use of Big Data throughout virtually every industry. [1]

4 ALGORITHMS

Algorithm 1 BnB($\{v_0, \ldots, v_{n-1}\}$): Find minimum cost Hamiltonian Cycle for euclidean distances

---

Data: $\{v_0, \ldots, v_{n-1}\}$ set of 2-D points
for all Unordered Pairs $\{i, j\}$ **do**
   Construct edge $e = (v_i, v_j, d_{ij})$
   Add e to list E of edges in increasing weight order: $E = E \cup \{e\}$
end for

Algorithm 2 2-Opt_HC($\{v_0, \ldots, v_{n-1}\}$): Approximate the minimum cost Hamiltonian Cycle for euclidean distances using a Hill Climbing local search algorithm with 2-Opt exchange Neighborhood Creation

---

Data: $\{v_0, \ldots, v_{n-1}\}$ set of 2-D points
for all Unordered Pairs $\{i, j\}$ **do**
    Construct edge $e = (v_i, v_j, d_{ij})$
    Add e to list E of edges in increasing weight order: $E = E \cup \{e\}$
end for
while Unassigned nodes in v **do**
    Assign Nodes to Route based on Greedy Nearest Neighbor implementation
end while
for $i = 1$ to length(Route Matrix) **do**
    for $j = i + 1$ to length(Route Matrix) **do**
        reverse array (route[i] to route[j]) and add it to newroute[i] to newroute[j]
        if cost(newroute) < cost(route) **then**
            route ← newroute
        end if
    end for
end for

Algorithm 3 Sim_Anneal($\{v_0, \ldots, v_{n-1}\}$): Approximate the minimum cost Hamiltonian Cycle for euclidean distances using a Hill Climbing local search algorithm with 2-Opt exchange Neighborhood Creation

---

Data: $\{v_0, \ldots, v_{n-1}\}$ set of 2-D points
Current_Route: $\{c_0, \ldots, c_{n-1}\}$ Set of Location Nodes denoting the current route for annealing
Best_Route: $\{b_0, \ldots, b_{n-1}\}$ Set of Location Nodes denoting the best route calculated so far for annealing
Temperature: T Current Annealing Temperature used
Cooling Ratio: α Ratio used to cool the temperature as Simulated Annealing is run
for all Unordered Pairs $\{i, j\}$ **do**
    Construct edge $e = (v_i, v_j, d_{ij})$
    Add e to list E of edges in increasing weight order: $E = E \cup \{e\}$
end for
while Unassigned nodes in v **do**
    Assign next node in route as the remaining node with the shortest distance between itself and the current node
end while
while Temperature ≥ Ending Temperature **do**
    Generate Random 2 Exchange Permutation
    if Current Solution Cost > Neighbor Route Cost **then**
        Update Current Route
    else
        Calculate Probability Using Current Temperature
        if Probability > Randomly Generated Probability **then**
            Update Current Route
        end if
    end if
    if Current Cost ≤ Best Cost **then**
        Update Best Route
    end if
end while

EMPIRICAL EVALUATION

The Qualified Runtime Distribution (QRTD) graphs for UMissouri show the major difference between the two local search algorithms, being that the Simulated Annealing algorithm not only performs better but is able to continue to increase its accuracy with a much higher probability as the algorithm continues to run. Figure 4 additionally shows that the Simulated Annealing algorithm is able to reach a good solution quickly, and then iterate on it initially relatively quickly. However, we see that as the solution moves closer to the optimal set the

Table 1: Approx Algorithm Performance Table

Instance	Time[s]	Solution Quality	Relative Error
Atlanta	0.00	2488307	0.2418
Berlin	0.02	10114	0.3410
Boston	0.00	1107063	0.2390
Champaign	0.02	64760	0.2302
Cincinnati	0.00	318227	0.1449
Denver	0.03	129206	0.2865
NYC	0.02	1927253	0.2393
Philadelphia	0.00	1697409	0.2159
Roanoke	0.27	796030	0.2145
SanFrancisco	0.03	1085013	0.3392
Toronto	0.06	1652074	0.4046
UKansasState	0.00	70318	0.1168
UMissouri	0.05	170427	0.2842

Table 2: BnB Algorithm Performance Table

Instance	Time[s]	Solution Quality	Relative Error
Atlanta	0.02	2003763	0.0000
Berlin	0.55	8087	0.0723
Boston	402.47	958728	0.0730
Champaign	42.53	60181	0.1432
Cincinnati	52.53	277952	0.0000
Denver	570.83	112944	0.1246
NYC	1.55	1783236	0.1467
Philadelphia	177.14	1482811	0.0622
Roanoke	541.27	757720	0.1560
SanFrancisco	6.59	857727	0.0587
Toronto	9.62	1218693	0.0362
UKansasState	22.58	62962	0.0000
UMissouri	119.11	150638	0.1351

Table 3: LS1 Algorithm Performance Table

Instance	Time[s]	Solution Quality	Relative Error
Atlanta	0.01	2003763	0.0000
Berlin	0.41	8087	0.0723
Boston	0.04	999953	0.1191
Champaign	0.54	61010	0.1589
Cincinnati	5.15	214510	0.2282
Denver	2.81	116743	0.1624
NYC	0.80	1783236	0.1467
Philadelphia	0.05	1498008	0.0731
Roanoke	18.00	783018	0.1946
SanFrancisco	2.55	857727	0.0587
Toronto	6.31	1218693	0.0362
UKansasState	2.09	42791	0.3204
UMissouri	1.81	153554	0.1571

Table 4: LS2 Algorithm Performance Table

Instance	Time[s]	Solution Quality	Relative Error
Atlanta	0.03	2003763	0.0000
Berlin	9.42	7712	0.0225
Boston	10.40	908113	0.0163
Champaign	10.75	53896	0.0238
Cincinnati	0.01	280282	0.0084
Denver	7.68	109965	0.0949
NYC	9.42	1666127	0.0714
Philadelphia	9.51	1396495	0.0004
Roanoke	7.18	757971	0.1564
SanFrancisco	9.30	845412	0.0435
Toronto	8.66	1212031	0.0305
UKansasState	0.00	63664	0.0111
UMissouri	14.63	147186	0.1091
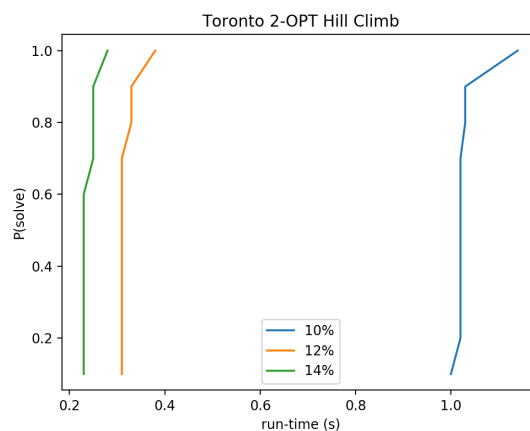
Figure 1: QRTD for the 2-OPT Hill Climb on the Toronto Dataset
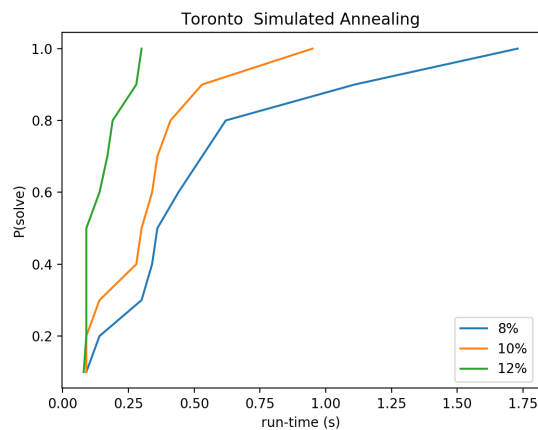
Figure 2: QRTD for the Simulated Annealing on the Toronto Dataset
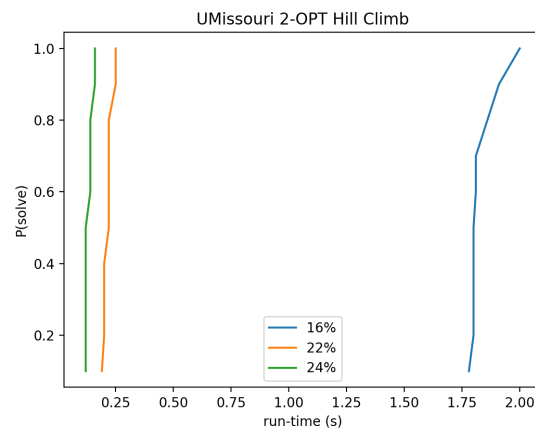
Figure 3: QRTD for the 2-OPT Hill Climb on the UMissouri Dataset
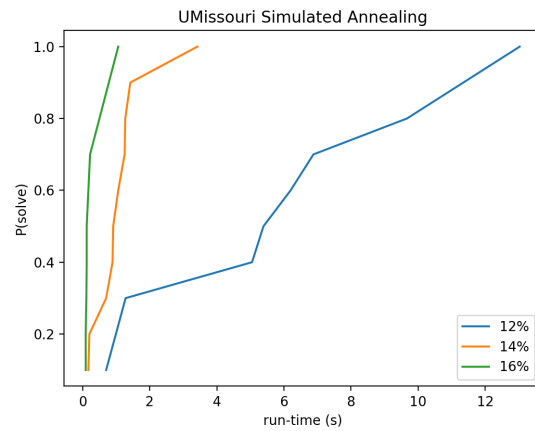
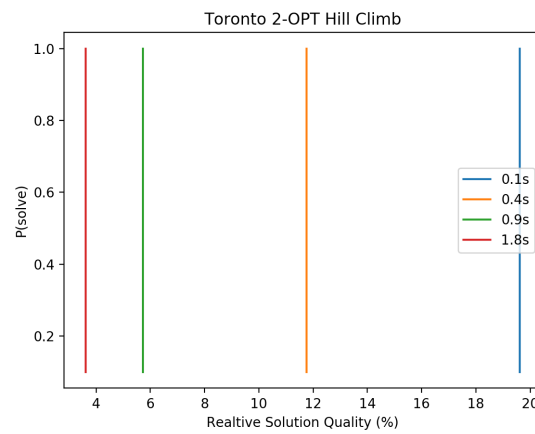**Figure 4: QRTD for Simulated Annealing on the UMissouri Dataset**

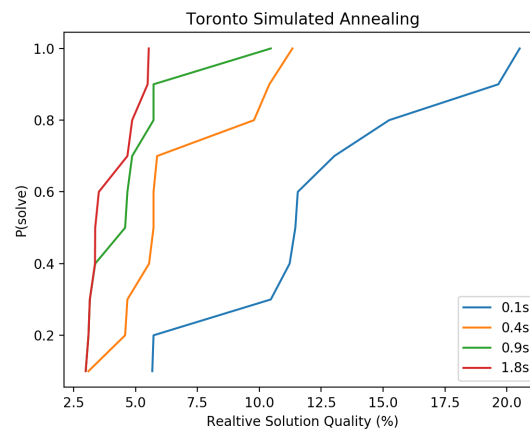**Figure 5: SQD for 2-OPT Hill Climb on the Toronto Dataset**

Figure 6: SQD for Simulated Annealing on the Toronto Dataset
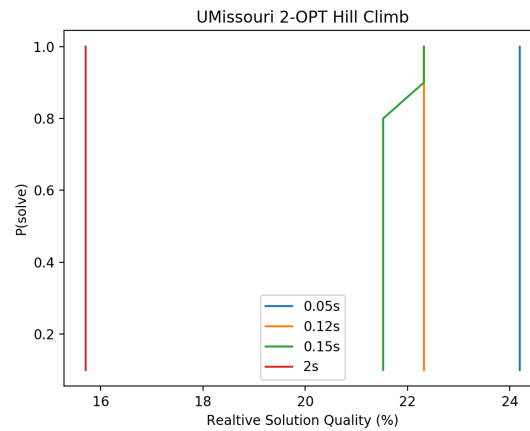

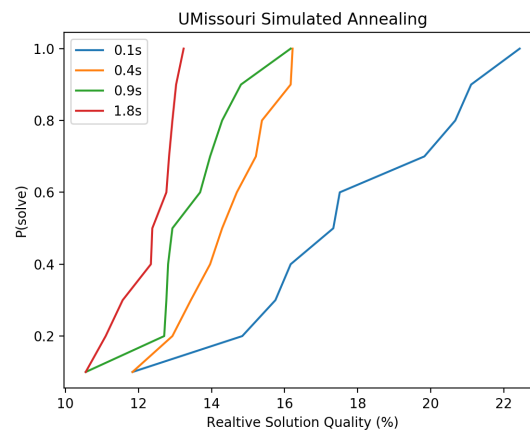**Figure 7: SQD for the 2-OPT Hill Climb on the UMissouri Dataset**

Figure 8: SQD for Simulated Annealing on the UMissouri Dataset

slope of the line begins to flatten out as the improvements take longer to make, something that intuitively makes sense as the set of possible solutions which are an improvement over the current solution decrease, while the search space remains stable.

The Solution Quality Distribution (SQD) graphs show the difference in the algorithms due to the highly random nature of the Simulated Annealing implementation in comparison to the 2-OPT Hill Climb implementation. When looking at Figure 5 and Figure 7 we observe that the solution qualities relative to time over all of the runs are almost always the same when the values are truncated to two decimal places. This is in contrast to the Simulated Annealing algorithms shown in Figure 6 and Figure 8 where the time is takes to reach a specified solution quality varies much more. Considering the different use-cases for an algorithm such as this there are situations where both cases are desireable, depending on whether speed or dependability are more important.
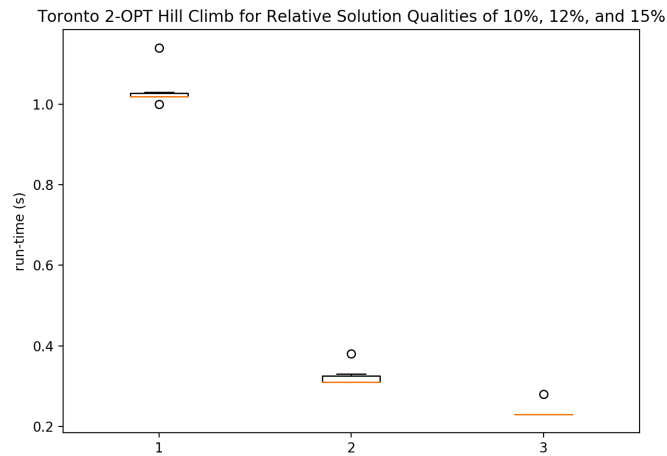

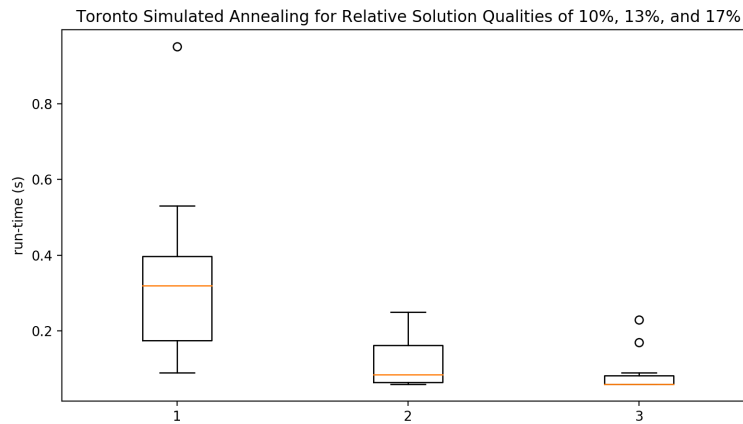**Figure 9: Box Plot for 2-OPT Hill Climb on the Toronto Dataset**

Figure 10: Box Plot for Simulated Annealing on the Toronto Dataset

The box plot detailed in Figure 10 is especially interesting with regards to the way which the run-times change as the error of the algorithm decreases. The first box plot shows the run-times to reach a relative error of 10%, while the third plot shows the run-times for a solution error of 17%. The lower error calculations not only have greater magnitude, but also a much higher variance which shows not only the obvious increase in execution time but the increase in run-time variability.
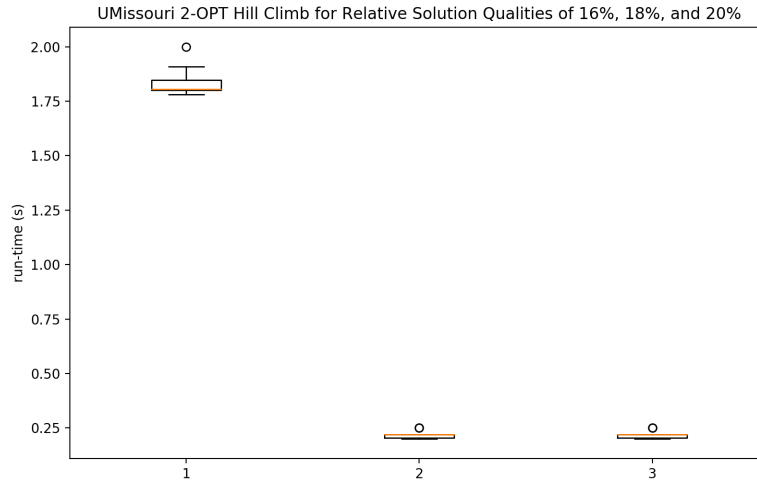
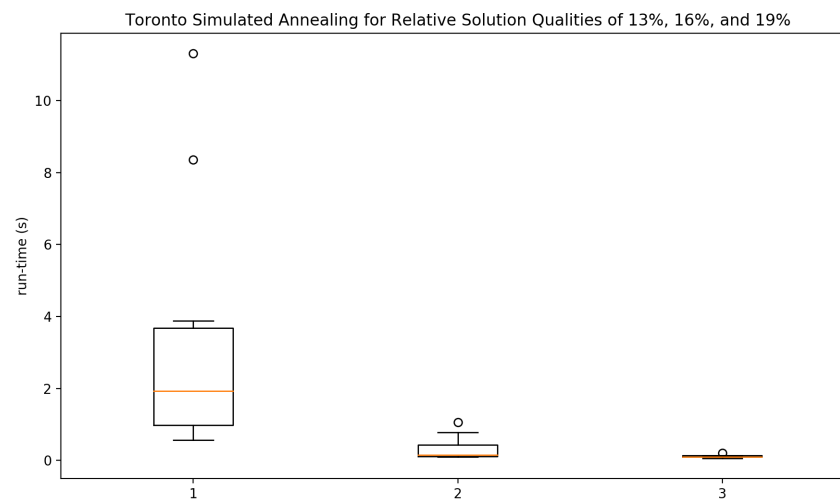**Figure 11: Box Plot for the 2-OPT Hill Climb on the UMissouri Dataset**

Figure 12: Box Plot for Simulated Annealing on the UMissouri Dataset

5   DISCUSSION

Branch-and-Bound
Approximation Algorithm
Local Search The 2-Opt Exchange and Simulated Annealing algorithms each performed relatively well for the majority of data-sets based on algorithm run-time versus performance. For each algorithm instance, a short algorithm runtime was able to produce results which were in most cases within 5% of the optimal solution. Each algorithm setup used a simple Greedy approximation algorithm as the initial path for local search. When testing different initial paths, the Greedy path proved to provide a good mix between accuracy and execution time as a strong initial solution was presented quickly by the algorithm, allowing more local search iterations per time-frame. The Simulated Annealing provided a clear benefit over the 2-opt exchange hill climbing setup however, which was due mainly to the ability of the algorithm to consider a broader neighborhood than the strict neighborhood that the 2-Opt exchange argument considered. To more clearly identify this, the α value was varied with a clear decrease in performance observed when the α value was deviated far from the eventual selection of .98 in either direction. This α value proved to have a reasonable affect on the annealing time-frame which allowed the algorithm to consider the entire search space thoroughly enough while still reaching a reasonable solution in a relatively short period of time. The 2-Opt algorithm struggled with a smaller neighborhood as only 2-Opt exchanges starting from all Greedy solutions were considered, and only those neighborhoods who provided a clear one to one improvement over the current best were used. The power of the Simulated Annealing setup becomes clear when looking at both of these local search algorithms as a similar 2-Opt Exchange neighborhood is considered in both, however the ability of Simulated Annealing to allow worse routes temporarily with the hope that they eventually lead to a more desireable solution proved to be key.

6   CONCLUSION

As is true with any engineering design, the type of implementation that should be used when looking to solve the TSP is heavily based on the type of results that are desired. All of the algorithms that were tested proved optimal for various situations depending on whether or not the driving factors for the implementation were speed or accuracy of the final solution in relation to the optimal. For example, the Branch-and-Bound algorithm proved to be reasonable when the dataset was relatively small, time was not a factor, and the desired outcome was to obtain a solution close or equal to the optimal. When the solution space ballooned the branch-and-bound did not provide as good of results compared to the extra time that it took to traverse the solution space, often becoming stuck traversing branches which in the end proved worse or not much better than previous solutions. For these larger cases both the Simulated Annealing and 2-OPT Hill Climb algorithms performed in many cases better than the Branch-and-Bound algorithm in considerably less time. Had the Branch-and-Bound algorithm been given hours to run instead of the designated 10 minutes, the solution returned likely would have been much closer to optimal than the Local Search algorithms were able to provide. However, when compute power and time is a constraint the Local Search algorithms proved to be the optimal method for evaluating the TSP algorithm. The Approximation algorithm in all cases returned with the greatest error, something which is to be expected based on the algorithm implementation. However, the extremely short runtimes make the approximation algorithm reasonable when looking to computer an initial solution, or when solution error is not the most important factor to consider execution time is prioritized.

REFERENCES

[1] ERICA KLARREICH. 2013. *Computer Scientists Take Road Less Traveled.* https://www.quantamagazine.org/computer-scientists-find-new-shortcuts-to-traveling-salesman-problem-20130129/
[2] Gilbert Laporte. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 2 (1992), 231âĂŞ247. https://doi.org/10.1016/0377-2217(92)90138-y