A Journey to Optimality: Deciphering the Traveling Salesman Problem

By: Faisal Baig, Neelkamal Bhuyan, Milad Heydari, Mehrdad Moradi, and Robert Quinn

Introduction

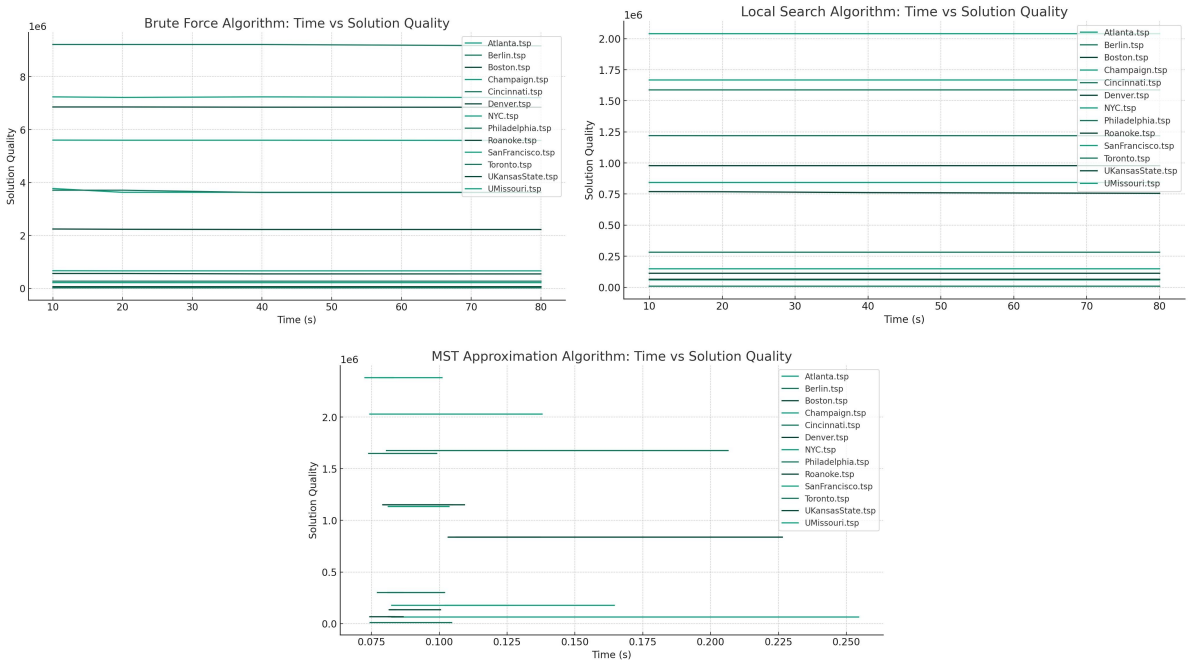
The Traveling Salesman Problem is a classic NP-Complete problem in computer science that involves determining the shortest possible path through a graph with weighted edges that visits each node exactly once and then returns to the start node [1]. The Traveling Salesman Problem has a number of important applications in science and logistics issues, from routing vehicles to DNA sequencing [2]. Given the importance of these applications to the function of society and the computational difficulty of the problem, knowing what approaches can most efficiently solve the Traveling Salesman Problem is crucial. The present work investigates the efficiency of three different algorithmic approaches to solving the Traveling Salesman Problem.

Methodology

A brute force search algorithm, a 2-approximation minimum spanning tree (MST) algorithm, and a local search with hill climbing algorithm were run on thirteen different datasets containing weighted graphs of city maps to determine their performance in solving the Traveling Salesman Problem. Each algorithm was run multiple times with time limits of 10, 20, 40, and 80 seconds, respectively.

The brute force search algorithm starts from a starting node, iterates through all possible permutations of paths that begin at that node, calculates the total distance of each path, keeps track of the path that produces the minimum cost, and terminates once the specified time limit is reached. The brute force search algorithm was run once for every combination of the thirteen datasets and four time limits, and the time limit, distance of the solution found (or solution quality), and whether or not a full tour was found was recorded for each run. The 2-approximation MST algorithm forms a MST by running Prim's algorithm on the graph, performs a pre-order traversal on the nodes from the graph to form an approximate optimal tour through the nodes, and computes the total distance of taking this tour. The 2-approximation algorithm was run once for every combination of the thirteen datasets and four time limits, and the total runtime, distance of the solution found (or solution quality), and the relative error of the solution quality with respect to the local search algorithms' solution quality was recorded for each run. The local search with hill climbing algorithm starts at a random node in the graph, performs a random walk of the graph to form an initial solution, and proceeds to use a hill climbing paradigm to try to minimize the distance of the solution by randomly swapping random neighboring nodes along the path until the specified time limit runs out. The local search algorithm was run with a new randomly chosen start node for 10 times for every combination of the thirteen datasets and four time limits. For each combination of the datasets and the time limits, the time limit and the average distance of the 10 solutions found (or solution quality) was recorded. Since the relative error of the solution calculated with respect to the local search solutions would be 1 in every case, the relative error calculation was omitted for the local search algorithm.

Results



The graphs above illustrate how each algorithm's solution quality changes with respect to the increase in time limit for every dataset (see Appendix for full data table). Overall, the change in the quality of all the solutions is minimal. For the brute force algorithm, two datasets (Cinicinnati, and UKansasState) exhibited no change in solution quality as the time limit increased, while the other datasets exhibited an average improvement (decrease) in solution quality of 1.1%. All 11 datasets exhibited no change in solution quality as the time limit for the 2-approximation MST algorithm. For the local search algorithm, Raonoke showed an improvement (decrease) in solution quality of 1.8%, Toronto showed an improvement (decrease) in solution quality of 0.05%, and all other datasets showed no change in solution quality as the time limit increased. Almost all of the algorithm runs exhibited a relative error of greater than 1 indicating that the Local Search algorithm produced the optimal result in the majority of cases, with the only two exceptions being when the brute force algorithm was run on Cincinnati and UKansasState. That being said, the 2-approximation MST algorithm always ran in under 1 second with consistently low relative errors in solution quality with respect to the Local Search solutions (never over ~1.37).

Conclusion

Overall, the Local Search with Hill Climbing approach produced the shortest solution to the Traveling Salesman Problem. However, the fact that these results were from an average of 10 runs makes this approach more computationally expensive than the other options. The 2-Approximation MST algorithm seems to strike a good balance between being computationally efficient and producing a decently optimal solution. The brute force search algorithm can find

optimal solutions in situations where the graph is small enough or the time limit is high enough. Additionally, higher time limits could allow the algorithms to find more optimal solutions. In the end, the trade off between computational efficiency and solution optimization will always be important to consider when deciding the best way to solve the Traveling Salesman Problem for a given application.

References

1. https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/
2. https://en.wikipedia.org/wiki/Travelling_salesman_problem

Appendix

Dataset	Time(s) - BF	Sol.Quality - BF	Full Tour - BF	RelError - BF	Time(s) - Approx	Sol.Quality - Approx	RelError - Approx	Time(s) - LS	Sol.Quality - LS
Atlanta.tsp	10	3775843	Yes	1.8509887220293484	0.078107834	2380448	1.1669400452765961	10	2039906
Atlanta.tsp	20	3630534	Yes	1.779755537755171	0.08323288	2380448	1.1669400452765961	20	2039906
Atlanta.tsp	40	3630534	Yes	1.779755537755171	0.072556973	2380448	1.1669400452765961	40	2039906
Atlanta.tsp	80	3630534	Yes	1.779755537755171	0.10103392601013184	2380448	1.1669400452765961	80	2039906
Berlin.tsp	10	19718	Yes	2.424443624738718	0.1045680046081543	10402	1.2789868437231033	10	8133
Berlin.tsp	20	19448	Yes	2.391245542850117	0.10368204116821289	10402	1.2789868437231033	20	8133
Berlin.tsp	40	19438	Yes	2.390015984	0.10386085510253906	10402	1.2789868437231033	40	8133
Berlin.tsp	80	19249	Yes	2.3667773269396286	0.074485064	10402	1.2789868437231033	80	8133
Boston.tsp	10	2244804	Yes	2.2918160220605275	0.10482096672058105	1150963	1.1750671524992164	10	979487
Boston.tsp	20	2234116	Yes	2.280904187600244	0.0826931	1150963	1.1750671524992164	20	979487
Boston.tsp	40	2227323	Yes	2.2739689245492793	0.079136133	1150963	1.1750671524992164	40	979487
Boston.tsp	80	2226767	Yes	2.2734012804662034	0.10924220085144043	1150963	1.1750671524992164	80	979487
Champaign.tsp	10	218074	Yes	3.7321627218428572	0.082438707	65712	1.1246085126046106	10	58431
Champaign.tsp	20	216440	Yes	3.704198114014821	0.087420225	65712	1.1246085126046106	20	58431
Champaign.tsp	40	215760	Yes	3.692560456	0.10299015045166016	65712	1.1246085126046106	40	58431
Champaign.tsp	80	216866	Yes	3.7114887645256798	0.2544987201690674	65712	1.1246085126046106	80	58431
Cincinnati.tsp	10	277952	Yes	0.9827341273404955	0.077218771	301216	1.064986914650712	10	282835.4
Cincinnati.tsp	20	277952	Yes	0.9827341273404955	0.10191011428833008	301216	1.064986914650712	20	282835.4
Cincinnati.tsp	40	277952	Yes	0.9827341273404955	0.080791235	301216	1.064986914650712	40	282835.4
Cincinnati.tsp	80	277952	Yes	0.9827341273404955	0.086536884	301216	1.064986914650712	80	282835.4
Denver.tsp	10	563620	Yes	5.027159612897472	0.083845854	134748	1.2018730767515498	10	112115
Denver.tsp	20	562770	Yes	5.01957811176024 65	0.081565142	134748	1.2018730767515498	20	112115
Denver.tsp	40	547974	Yes	4.887606475493913	0.096750975	134748	1.2018730767515498	40	112115
Denver.tsp	80	546699	Yes	4.8762342237880745	0.10042285919189453	134748	1.2018730767515498	80	112115
NYC.tsp	10	7239945	Yes	4.342436725298005	0.074445963	2027107	1.2158357394854054	10	1667254
NYC.tsp	20	7218153	Yes	4.329366131375303	0.081619978	2027107	1.2158357394854054	20	1667254
NYC.tsp	40	7239945	Yes	4.342436725298005	0.13786530494689941	2027107	1.2158357394854054	40	1667254
NYC.tsp	80	7210976	Yes	4.325061448345603	0.12978577613830566	2027107	1.2158357394854054	80	1667254
Philadelphia.tsp	10	3710782	Yes	2.3360171934922582	0.079373121	1646249	1.0363492031516366	10	1588508
Philadelphia.tsp	20	3710782	Yes	2.3360171934922582	0.082762003	1646249	1.0363492031516366	20	1588508
Philadelphia.tsp	40	3624919	Yes	2.281964585636333	0.099001646	1646249	1.0363492031516366	40	1588508
Philadelphia.tsp	80	3624919	Yes	2.281964585636333	0.073941946	1646249	1.0363492031516366	80	1588508
Roanoke.tsp	10	6857992	Yes	8.909942952	0.10620903968811035	838282	1.0891008326609943	10	769701
Roanoke.tsp	20	6857992	Yes	8.922241881765196	0.22646760940551758	838282	1.0906041840133223	20	768640
Roanoke.tsp	40	6849948	Yes	8.993182590694309	0.10325884819030762	838282	1.1005664697735524	40	761682.3
Roanoke.tsp	80	6847051	Yes	9.059654076	0.13705801963806152	838282	1.1091702016740455	80	755774
SanFrancisco.tsp	10	5604793	Yes	6.654374121423958	0.10326814651489258	1134989	1.347532626040044	10	842272
SanFrancisco.tsp	20	5600573	Yes	6.649363863455036	0.08119297	1134989	1.347532626040044	20	842272
SanFrancisco.tsp	40	5600573	Yes	6.649363863455036	0.10099673271179199	1134989	1.347532626040044	40	842272
SanFrancisco.tsp	80	5591805	Yes	6.638953924622925	0.10360288619995117	1134989	1.347532626040044	80	842272
Toronto.tsp	10	9219353	Yes	7.555425434313791	0.098344088	1675108	1.3727811038825073	10	1220229.5
Toronto.tsp	20	9219353	Yes	7.5586831095685865	0.17255282402038574	1675108	1.3733730063599057	20	1219703.6
Toronto.tsp	40	9219353	Yes	7.557171927480486	0.2063918113708496	1675108	1.373098432514514	40	1219947.5
Toronto.tsp	80	9168496	Yes	7.518220324910916	0.080591679	1675108	1.3735983537562624	80	1219503.5
UKansasState.tsp	10	62962	Yes	1	0.086728334	68090	1.0814459515263175	10	62962
UKansasState.tsp	20	62962	Yes	1	0.081019163	68090	1.0814459515263175	20	62962
UKansasState.tsp	40	62962	Yes	1	0.083821058	68090	1.0814459515263175	40	62962
UKansasState.tsp	80	62962	Yes	1	0.074365854	68090	1.0814459515263175	80	62962
UMissouri.tsp	10	668324	Yes	4.498256760940676	0.082489014	178249	1.1997321200209996	10	148574
UMissouri.tsp	20	662953	Yes	4.462106425081104	0.1257152557373047	178249	1.1997321200209996	20	148574
UMissouri.tsp	40	662935	Yes	4.461985273331807	0.10200810432434082	178249	1.1997321200209996	40	148574
UMissouri.tsp	80	662935	Yes	4.461985273331807	0.16450715065002441	178249	1.1997321200209996	80	148574