2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)

A Greedy-Genetic Local-Search Heuristic for the Traveling Salesman Problem

Mohammad Harun Rashid
Seidenberg School of CSIS
Pace University
New York, NY
Email: mr29963n@pace.edu

Miguel A. Mosteiro
Computer Science Dept.
Pace University
New York, NY
Email: mmosteiro@pace.edu

Abstract

The Travelling Salesman Problem (TSP) is one of the typical combinatorial optimization problems that is easy to describe but hard to solve. In this work, we present a novel solution that integrates a genetic algorithm, local-search heuristics, and a greedy algorithm. For the genetic algorithm we keep the evolutionary technique to generate children from parents, which uses operators like mutation, selection of the most fitted element, and crossover, but the latter is enhanced with a local-search heuristic. We also use the local search heuristic for its strong climbing ability, as well as to find local optima efficiently in the TSP space. The greedy algorithm is used to generate new greedy children from parents. The experimental evaluation shows that the optimization algorithm presented provides higher quality solutions for TSP with respect to previous genetic algorithms, within reasonable computational time.

Keywords: travelling salesman problem, TSP, genetic algorithms, local search, tabu search.

I. Introduction

In the Travelling Salesman Problem (TSP), a salesman has to visit a number $n > 0$ of cities, with the restriction that each city must be visited exactly once, and return to the origin city. The problem is to find the order in which the cities must be visited to minimize the path length as compared to other possible routes. This shortest route is called a *Hamiltonian circuit* in the context of graph theory, where the graph is defined by the set of pairs of connected cities. For small instances of the problem the optimal solution can be found by exhaustive search. But as the number of cities increases, the number of possible paths increases exponentially, as the number of possible permutations is $n!$. Thus, computing the cost of all possible paths would take too much time. Moreover, the TSP in general graphs is well-known to be NP-hard, and even in the metric and Euclidean versions [1]–[3]. Thus, approximation algorithms are studied.

TSP has a wealth of real-world applications in many areas, such as computer networking, electronic maps, traffic induction, electrical wiring, VLSI layout, etc. Thus, it is often regarded as a problem of paramount importance, beyond theoretical interest. TSP is closely related to parallel and distributed computing in different ways. On one hand, aiming to speed up the computation, the parallel implementation of TSP solutions has been studied [4], [5], and also for settings where TSP instances are available in batches [6]. More relevant for our work, good approximations for TSP reduce message complexity for various distributed computing problems in communication networks with weighted links, such as millimeter-wave [7], [8] communication. In this sense the TSP algorithm presented here entails a solution for scheduling and resource management. For a thorough review of the state of the art on TSP approximation algorithms we refer the reader to [9].

Our approach makes use of a genetic algorithm, as frequently used for TSP. Genetic algorithms entail optimization techniques that use special operators such as selection, reproduction and mutation to solve difficult problems, generating "children" solutions from "parent" ones in an evolutionary process. We combine these techniques with a local search heuristic and a greedy algorithm. The local search heuristic is responsible for finding local optima in the TSP space efficiently, and the greedy algorithm is responsible for generating new greedy children from parents. All in all, our approach incorporates a genetic algorithm, a local search method, and a greedy algorithm aiming to obtain an effective heuristic approach to find an improved solution for TSP. Our experimental evaluation shows for random placement of cities an improvement over previous work of up to 17% shorter paths, with an average of approximately 12%. Our results provide also insight on the potential approximation improvement that could be obtained even on a distributed system. Indeed, a potential follow-up of this work is to study settings with clustered cities, as within a cluster our greedy approach could be implemented to decide the next city to visit.

II. Background

a) Genetic Algorithms: A genetic algorithm [10] is an optimization technique for solving combinatorial problems that is based on natural selection, which drives biological evolution. A genetic algorithm randomly selects individuals from the current population at each step to be parents and uses the parents to produce the children for the next generation.

0-7695-6329-5/17/31.00 ©2017 IEEE
DOI 10.1109/ISPA/IUCC.2017.00132

This algorithm finds an optimal solution after a number of successive generations by repeatedly modifying a population of individual solutions. We can implement genetic algorithms to solve various optimization problems that are not well suited for standard optimization. A genetic algorithm consists of these steps: 1) encoding, 2) evaluation, 3) selection, 4) crossover, 5) mutation, and 6) decoding. A basic genetic algorithm is as follows.

	Genetic Algorithm Structure
Step-1:	Initialize population with random solutions.
Step-2:	Evaluate each candidate.
Step-3:	Repeat (iterations).
	a) Select parents.
	b) Recombine pairs of parents.
	c) Mutate the resulting children.
	d) Evaluate children.
	e) Select individuals for next generation.
	Until Terminating condition is satisfied.

b) Local Search Heuristics: Local search [11] is an iterative heuristic algorithm for solving hard combinatorial optimization problems. A local search algorithm moves from one solution to another solution according to some neighborhood structure. Local search iterates through the possible candidate solutions until a solution considered optimal is found or a time bound is elapsed. Hill climbing, tabu search, Simulated annealing etc. are some of the local search heuristics. A local search heuristic starts with a random initial solution as its current solution and then goes through loop iterations. During each iteration, a random neighbor of the current solution is generated. If the neighbor improves the solution metric, then the neighbor becomes the new current solution for the next iteration and the loop repeats. The process will terminate when some termination criteria is met. Tabu search relaxes its basic rule and enhances the performance of local search. When no improving move is available and the search is stuck at a strict local minimum. Tabu search still accepts the worsening move. It is often used for problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time and also, when the search space is discrete (e.g., all tours that visit a given set of cities).

c) Greedy Algorithms: A greedy algorithm always makes the choice that seems to be the best at that moment. It makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution. If we assume that we have an objective function that needs to be maximized or minimized at a given point, a greedy algorithm makes choices at each step to ensure that the objective function is optimized at that point, without considerations on the impact in future decisions, and without reversing the decision later. For instance, a greedy strategy for TSP is the following heuristic: "At each stage visit an unvisited city nearest to the current city". This heuristic may not find an optimal solution, but terminates in a polynomial number of steps. However, it has been shown [12] that greedy techniques alone do not provide efficient solutions for TSP. In general, greedy algorithms have five components:

- A candidate set, from which a solution is created.
- A selection function, which chooses the best candidate to be added to the solution.
- A feasibility function, that is used to determine if a candidate can be used to contribute to a solution.
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution.

Greedy algorithms produce good solutions on some mathematical problems. We can make whatever greedy choice seems best at the moment and then solve the sub problems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub problem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one.

III. Problem Statement

The *Traveling Saleman Problem* (TSP) is the problem of finding the shortest route through a collection of cities, visiting each city exactly once and returning to the initial city. As the number of cities increases, the possible number of paths increase exponentially with the number of cities.

IV. Related Work:

The literature on TSP is vast and its comprehensive overview is beyond the scope of this work. We focus in this section on heuristics that use genetic algorithms.

Some recent works [13]–[16] are focused on genetic algorithms, without introducing local/neighborhood searches. Lin et al. [17] proposed a novel heuristic Hybrid Genetic Algorithm (HGA), which is a mechanism of combining GA with the local search strategy, for solving TSP. The proposed HGA incorporates the local search strategy into GA by the novel updating strategy which is the combination of the elitist choice strategy, the local search crossover operator and the double-bridge random mutation. It takes full advantage of exploration ability of GA and exploitation capability of the local search method to improve the quality of the optimum or sub-optimum solutions with reasonable time-consuming. The method shows some improvements in finding the solution of TSP.

Wang et al. [18] provided an improved algorithm called a neighborhood expansion tabu search algorithm based on genetic factors (NETS) to solve TSP. The algorithm keeps the traditional tabu algorithms neighborhood, ensure the algorithms strong climbing ability and go to the local optimization. At the same time, introduce the genetic algorithms genetic factor (crossover factor and variation factor) to develop new search space for bounded domain. It can avoid the defects of the alternate search. The results show that this optimization algorithm has improved in the target of target value, convergence and compared with traditional tabu algorithm and genetic algorithm.

V. Our Contribution

In this work we present an algorithm for TSP that integrates a genetic algorithm with local-search heuristics and a greedy algorithm. In our approach we keep the evolutionary technique of genetic algorithms using operators like mutation, crossover and the selection of the most fitted element, but the crossover operator is enhanced with a local search heuristic. We also use the local search heuristic for its strong climbing ability as well as to find local optima in the TSP space efficiently. In our greedy approach, a chromosome (travel path) is rearranged according to nearest-neighbor-first greedy strategy. Our experimental evaluation shows that our algorithm finds better solutions than previous local-search algorithms in [17], [18].

VI. Design and methods

As described earlier, our approach integrates local search heuristics and a greedy algorithm into a genetic algorithm. Specifically, the crossover operator in genetic algorithms is enhanced with a local search heuristic, while keeping operators like mutation, crossover and the selection of the most fitted element. The local search heuristic is used to take advantage of its strong climbing ability while finding local optima in the TSP. The greedy algorithm is used to generate new greedy children from parents. Following the general genetic framework detailed in Section II, our main TSP heuristic is detailed in Figure 1.

Our greedy approach in Step 3b) in Figure 1 is detailed in Figure 2. In brief, the idea is to rearrange a chromosome (i.e. a travel path) according to nearest neighbor first greedy strategy. For instance, consider the following example on 20 cities labeled arbitrarily with consecutive integers $1, 2, 3, \ldots, 20$. Let a parent path in the genetic evolution be the following.

```
1 10 7 5 11 18 14 2 6 19
                20 13 4 8 16 9 3 17 15 12 1
```

To generate a greedy child of this path, we select the first 10 cities from the parent path, and starting from the first one we rearrange the next 9 greedily. That is, we start a greedy child as

```
1 - - - - - - - - -
                20 13 4 8 16 9 3 17 15 12 1
```

Then, we find a city which is the nearest neighbor of city 1 out of the remaning 9 cities. Assume such city is 19. Then we get the following.

```
1 19 - - - - - - - -
                20 13 4 8 16 9 3 17 15 12 1
```

Now we apply the approach recursively to find the nearest neighbor of city 19 among the 8 remaining cities. Should it be 11 we get.

```
1 19 11 - - - - - - -
                20 13 4 8 16 9 3 17 15 12 1
```

We continue such process until all the 10 cities have been rearranged.

We have also developed a simulator with a graphical interface to visualize the paths obtained in our experiments. An examples of such rendering is given in Figure 3.

VII. Experimental Validation

To evaluate our heuristic in practice, we carried out experiments of the algorithms presented in the previous section in comparison with previous work.

We have used C#, WPF (Windows Presentation Foundation), Microsoft .NET Framework and Visual Studio.Net to create a visual simulator (cf. Figure 3). Our simulator calculates the total length obtained by our TSP algorithm based on a genetic algorithm combined with local search and a greedy algorithm. For comparison, we also evaluated experimentally the path length obtained by using a genetic algorithm with local search, without combining with a greedy strategy for children generation. Specifically, we evaluated the algorithms in [17], [18]. The heuristic approach followed in both works is similar: to change the genetic operators randomly, with some neighborhood search improvement. For clarity, we present only the comparison with [18]. Similar results may be obtained for [17]. Our visual simulator also displays the results obtained for different numbers of iterations.

We describe now our results for $n = 50$ cities. Similar results were obtained for $n = 20$ cities. The corresponding figures are omitted due to space restrictions. We have evaluated the TSP algorithms for $n = 20$ cities and $n = 50$ cities (cf. Fig 3 and 4). For our TSP algorithm, we greedily rearranged a subset of $k = 10$ cities according to the nearest-neighbor-first policy. The resulting path lengths were plotted for different iterations of the algorithm to expose the tradeoff between accuracy of the solution and computation time. It can be seen in Figure 3 an execution for $n = 50$ cities running the algorithms up to almost 200 iterations.

We also include the data points in Table I, and the corresponding plot in Figure 4.

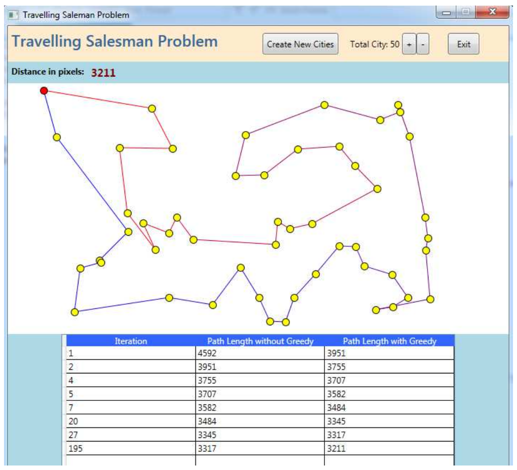
To simplify the development of our simulator, we produced inputs for Euclidean-TSP. That is, the cities are located in the plane, the distance among cities is the Euclidean distance, and all cities are reachable from each city. Although Euclidean-TSP is a particular case of TSP in general graphs, it is also known to be NP-hard [1], [2]. The input was then generated as follows. For a given number of cities n, the cities were placed uniformly at random in a grid of 350 rows and 600 columns. Then, the path lengths, using the rows and columns as units of Euclidean distance, obtained after various numbers of iterations were collected. The results obtained are discussed in the following section.

```
        TSP Greedy-Genetic Algorithm
Step 1: Generate a random initial population of chromosomes (travel path).
Step 2: Calculate fitness (path length) of every chromosome of initial population.
Step 3: Repeat (iterations).
          a) Apply local search heuristic to select parents from initial population for cross over.
          b) Apply a greedy algorithm to generate greedy children from parents.
          c) Calculate fitness of all newly generated children and add newly generated children
          in initial population.
          d) Sort combined population initial population + new children by path length and
          select best chromosomes (travel path) for next population.
          e) Set initial population = next population.
        Until Terminating condition is satisfied (for given number of generations).
```

Fig. 1. Main TSP Greedy-Genetic Algorithm

```
        Greedy-Children Algorithm
        input: an array parentPath containing $n$ cities and an integer $k \leq n$:
Step 1: Copy parentPath[1] into position 1 of a new array childPath of size $n$
Step 2: Copy parentPath[$2, \ldots, k$] into a new list prefix.
Step 3: Copy parentPath[$k+1, \ldots, n$] into childPath[$k+1, \ldots, n$]
Step 4: Set currentCity = parentPath[1]
Step 5: For $i = 2$ to $k$:
          a) Extract from prefix the city nextCity at minimum distance from currentCity.
          b) Store nextCity in childPath[$i$].
          c) Set currentCity = nextCity.
Step 6: Return childPath.
```

Fig. 2. Implementation of Step 3b) in Figure 1.



Distance in pixels: 3211

Iteration	Path Length without Greedy	Path Length with Greedy
1	4592	3951
2	3951	3755
4	3755	3707
5	3707	3582
7	3582	3484
20	3484	3345
27	3345	3317
195	3317	3211

Fig. 3. Results for $n = 50$ and $k = 10$ up to 195 iterations.

Iteration	Path length [18]	Path length (this work)
1	4592	3951
2	3951	3755
4	3755	3707
5	3707	3582
7	3582	3484
20	3484	3345
27	3345	3317
195	3317	3211

TABLE I
PATH LENGTH FOR $n = 50$ AND $k = 10$.

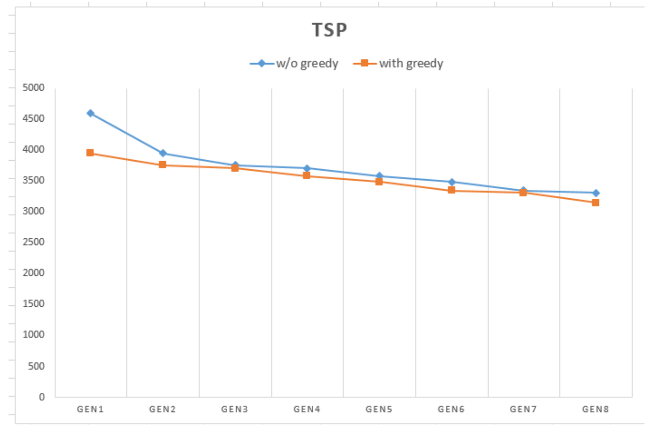


Fig. 4. Path length comparison of our TSP algorithm (with greedy) and the algorithm in [18] (without greedy) for $n = 50$ and $k = 10$.

VIII. Discussion of Results

The results obtained should be interpreted as follows. The algorithms evaluated are all based on local search heuristics. I.e., if there is no better solution found through neighborhood search, the initial solution will always be the last best solution. In the first iteration, our greedy approach finds a path length that is smaller than the solution obtained by [18], which is stored as the best solution so far. In the next iteration the algorithm in [18] could not find a better solution, leaving the best solution as is. However, our greedy approach finds a smaller path length in the next few iterations updating the best solution so far. In our simulator both algorithms share the same global variable storing the best path so far. Thus, the path length in a given row for the algorithm in [18] is the same as the path length obtained by our TSP algorithm in the previous row, as shown in Tables **??** and I. In summary, for each row, we observe the improvement obtained applying the greedy approach after some additional iterations whereas previous work based on local search is not able to improve further.

To observe the behavior of our TSP algorithm through multiple executions, we also include in Table II the path lengths obtained for $n = 20$ cities and 200 iterations. For each of these executions, the input was generated at random as explained above. It can be seen that our consistently obtains a shorter path length, with an improvement of 11.86% on average over these 10 executions.

Path length [18]	Path length (this work)	Improvement
2914	2604	10.64%
2322	2005	13.65%
2544	2333	8.29%
3072	2691	12.40%
2801	2585	7.71%
2622	2259	13.84%
2605	2358	9.48%
2559	2198	14.11%
2971	2638	11.21%
2212	1831	17.22%

TABLE II
PATH LENGTHS OBTAINED IN 10 EXECUTIONS, EACH OF 200 ITERATIONS, FOR $n = 20$ AND $k = 10$.

IX. CONCLUSIONS AND FUTURE WORK

In this work, we have presented a novel TSP algorithm integrating a greedy approach into a genetic algorithm with local-search. The experimental results presented expose the improvement achieved by our TSP algorithm over previous work as detailed in the previous section. All in all, the results presented show path lengths up to 17% shorter, with an average of approximately 12%. This improvement is due to the integration of the greedy approach.

With respect to future work, it would be interesting to explore alternatives that allow to make decisions based only on local (geographically speaking) information. On one hand, it would speed up the algorithm on settings where each city only has a few neighboring cities. On the other hand, having to process only local information would facilitate the distributed computation of the path. The greedy approach seems particularly well suited for such algorithm on clustered cities, as a cluster of more than k neighboring cities could easily compute the rearrangement according to nearest-neighbor-first policy.

REFERENCES

[1] M. R. Garey, R. L. Graham, and D. S. Johnson, "Some np-complete geometric problems," in *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '76. New York, NY, USA: ACM, 1976, pp. 10–22. [Online]. Available: http://doi.acm.org/10.1145/800113.803626

[2] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237 – 244, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397577900123

[3] R. M. Karp, "Reducibility among combinatorial problems," in *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, 1972, pp. 85–103. [Online]. Available: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf

[4] T. Fischer and P. Merz, "A distributed chained lin-kernighan algorithm for tsp problems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, ser. IPDPS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 16.2–. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2005.18

[5] N. Lai and B. P. Miller, "The traveling salesman problem: The development of a distributed computation," in *International Conference on Parallel Processing, ICPP'86, University Park, PA, USA, August 1986.*, 1986, pp. 417–420.

[6] S. Ozden, A. Smith, and K. Gue, "Solving large batches of traveling salesman problems with parallel and distributed computing," *Computers & Operations Research*, vol. 85, pp. 87 – 96, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054817300825

[7] W. Feng, Y. Li, D. Jin, L. Su, and S. Chen, "Millimetre-wave backhaul for 5g networks: Challenges and solutions," *Sensors*, vol. 16, no. 6, Jun. 2016.

[8] T. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. Wong, J. Schulz, M. Samimi, and F. Gutierrez, "Millimeter wave mobile communications for 5g cellular: It will work!" *IEEE Access*, pp. 335 – 349, May 2013.

[9] C. Rego, D. Gamboa, F. Glover, and C. Osterman, "Traveling salesman problem heuristics: Leading methods, implementations and latest advances," *European Journal of Operational Research*, vol. 211, no. 3, pp. 427 – 441, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221710006065

[10] M. Mitchell, *An introduction to genetic algorithms*. MIT Press, 1998.

[11] J. Hromkovic, *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics, Second Edition*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. [Online]. Available: https://doi.org/10.1007/978-3-662-05269-3

[12] G. Gutin, A. Yeo, and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp," *Discrete Applied Mathematics*, vol. 117, no. 1, pp. 81 – 86, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X01001950

[13] H. H. Chieng and N. Wahid, "A performance comparison of genetic algorithm's mutation operators in n-cities open loop travelling salesman problem," in *Recent Advances on Soft Computing and Data Mining - Proceedings of The First International Conference on Soft Computing and Data Mining (SCDM-2014) Universiti Tun Hussein Onn Malaysia, Johor, Malaysia, June 16th-18th, 2014*, 2014, pp. 89–97. [Online]. Available: https://doi.org/10.1007/978-3-319-07692-8_9

[14] B. Zanaj and E. Zanaj, "Review of traveling salesman problem for the genetic algorithms," *Journal of Information Sciences and Computing Technologies*, vol. 5, no. 3, pp. 534–545, 2016. [Online]. Available: http://scitecresearch.com/journals/index.php/jisct/article/view/829

[15] C. Loyola, M. Sepúlveda, M. Solar, P. Lopez, and V. Parada, "Automatic design of algorithms for the traveling salesman problem," *Cogent Engineering*, vol. 3, no. 1, p. 1255165, 2016. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/23311916.2016.1255165

[16] C. Contreras-Bolton and V. Parada, "Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem," *PLOS ONE*, vol. 10, no. 9, pp. 1–25, 09 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0137724

[17] B. Lin, X. Sun, and S. Salous, "Solving travelling salesman problem with an improved hybrid genetic algorithm," *Journal of Computer and Communications*, vol. 4, pp. 98–106, 2016.

[18] D. Wang, H. Xiong, and D. Fang, "A neighborhood expansion tabu search algorithm based on genetic factors," *Open Journal of Social Sciences*, vol. 4, no. 3, pp. 303–308, 2016.