Scientific Research Publishing

Journal of Applied Mathematics and Physics, 2024, 12, 1557-1570
https://www.scirp.org/journal/jamp
ISSN Online: 2327-4379
ISSN Print: 2327-4352

On the "Onion Husk" Algorithm for Approximate Solution of the Traveling Salesman Problem

Mikhail E. Abramyan[1,2], Nikolai I. Krainiukov[1], Boris F. Melnikov[1]

[1]Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU-BIT University, Shenzhen, China
[2]Department of Algebra and Discrete Mathematics, Southern Federal University, Rostov-on-Don, Russian Federation
Email: m-abramyan@yandex.ru, n.krainiukov@smbu.edu.cn, bormel@mail.ru

How to cite this paper: Abramyan, M.E., Krainiukov, N.I. and Melnikov, B.F. (2024) On the "Onion Husk" Algorithm for Approximate Solution of the Traveling Salesman Problem. *Journal of Applied Mathematics and Physics*, **12**, 1557-1570.
https://doi.org/10.4236/jamp.2024.124095

Received: February 20, 2024
Accepted: April 27, 2024
Published: April 30, 2024

Abstract

The paper describes some implementation aspects of an algorithm for approximate solution of the traveling salesman problem based on the construction of convex closed contours on the initial set of points ("cities") and their subsequent combination into a closed path (the so-called contour algorithm or "onion husk" algorithm). A number of heuristics related to the different stages of the algorithm are considered, and various variants of the algorithm based on these heuristics are analyzed. Sets of randomly generated points of different sizes (from 4 to 90 and from 500 to 10,000) were used to test the algorithms. The numerical results obtained are compared with the results of two well-known combinatorial optimization algorithms, namely the algorithm based on the branch and bound method and the simulated annealing algorithm.

Keywords

Branch and Bound Method, Contour Algorithm, "Onion Husk" Algorithm, Simulated Annealing Method, Traveling Salesman Problem

1. Introduction

Combinatorial optimization problems and methods for solving them are constantly evolving and improving; one such problem is the traveling salesman problem (TSP), which belongs to classical NP-complete problems [1]. It should be noted that the combinatorial optimization methods used to solve this problem can be used more widely—for optimization, search and processing of complex data and other combinatorial objects.

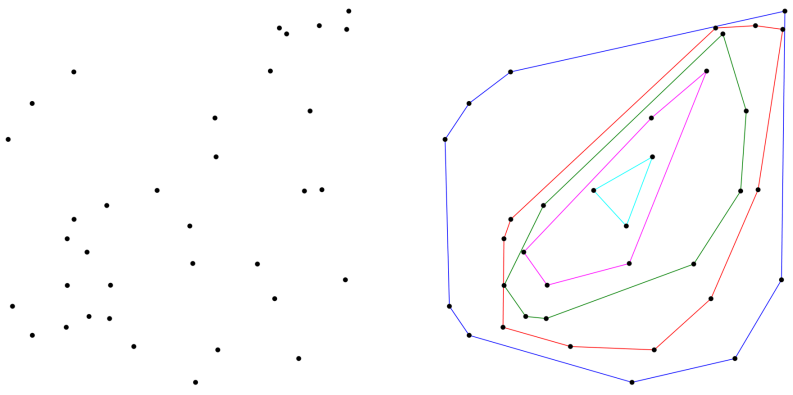
A large number of numerical methods for approximate solutions have been developed for the traveling salesman problem [2]. We discuss various aspects of the implementation of an algorithm that has received relatively little attention in the literature. This is the *contour algorithm*, or *"onion husk" algorithm*, which can be applied to approximate the solution of the so-called *geometric TSP* for which not only the distance matrix, but also the points themselves ("cities") in the plane are given. The contour algorithm is based on the construction of nested convex contours connecting the initial points (**Figure 1**) and the subsequent integration of these contours into a final closed path. The advantage of this algorithm is its high speed, while the obvious disadvantage is the approximate nature of the solution. At the same time, by applying additional heuristics at the main stage of the algorithm—merging the initial convex contours into the final closed path—it is possible to improve the results obtained.

2. Construction of a Set of Convex Contours and Related Cutoff Heuristics

We start with brief discussion of the ways of implementing the first stage of the contour algorithm where a family of nested convex contours is constructed. The problem of constructing a convex hull is a classical problem in computational geometry, and there are a number of efficient algorithms for its solution, in particular the Graham and Jarvis algorithms.

The *Graham algorithm* consists of two stages. At the initial stage (preprocessing), the initial set of points is sorted by a certain characteristic (increasing angle formed by the given point and some fixed point). At the main stage (traversal), the sorted points are bypassed and the required convex hull is constructed. Due to the preprocessing step, the complexity of this algorithm is $O(n \log n)$, where n is the number of points.

The *Jarvis algorithm*, also called the *"gift wrapping" algorithm*, also consists of two stages. In the first stage (preprocessing), some edge point of the set (e.g. the



(a) (b)

Figure 1. Set of points (a) and their union into convex contours (b).

point with minimum abscissa min (X)) is chosen; this point is obviously included in the convex hull. In the second stage (traversal), the remaining points of the convex hull are successively constructed by analyzing the angles formed by one or two last obtained points of the hull and all points of the original set. Thus, the complexity of this algorithm is $O(nq)$, where q is the number of points in the constructed convex hull (when finding each of the q points of the convex hull, all n points of the original set are processed, see **Figure 2**).

To implement the first stage of the contour algorithm described in this paper, the Jarvis algorithm was chosen, and two heuristics were considered to speed up this algorithm. These heuristics cut off a part of the source points that are guaranteed not to be included in the convex hull. For this purpose, the set of edge points of the initial set is defined, after which all points of the initial set are analyzed and those that lie strictly inside the convex hull constructed by the edge are discarded. Thus, the implementation of this heuristic is a generalization of the preprocessing stage of the Jarvis algorithm, in which several points are found instead of one edge point, after which an additional cutoff is performed. Obviously, the complexity of this stage is $O(kn)$, where k is the number of edge points.

In the simplest version of this heuristic (Cutoff4), a set of four edge points is selected: with minimum and maximum abscissa and ordinate (min (X), max (X), min (Y), max (Y)). In special cases, two of these points may coincide, resulting in a set of three points, but this will not affect the subsequent stages of the algorithm (although it will result in cutting off fewer points). **Figure 3** shows the result of this heuristic for a set of 90 points. The edge points are highlighted with black circles; their number is 4. The cutoff points are marked with red circles; their number is 51. The remaining 35 points, which are likely to be included in

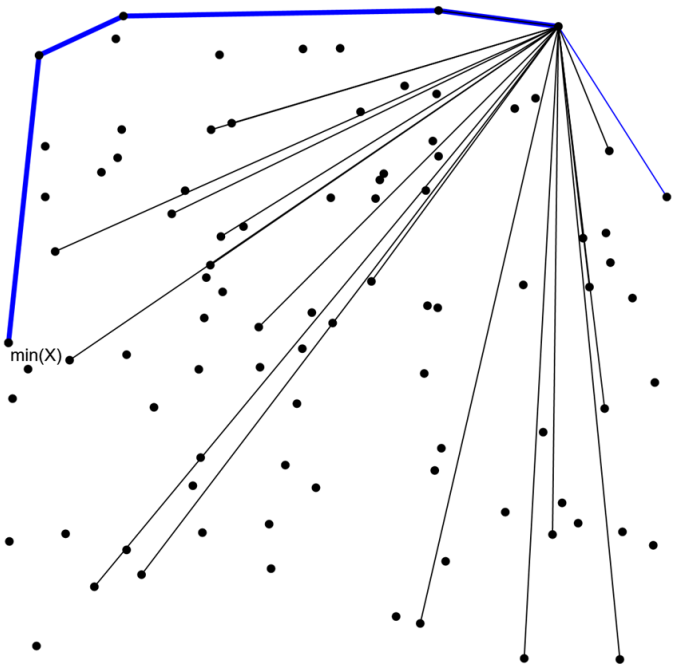


Figure 2. The main stage of the Jarvis algorithm.

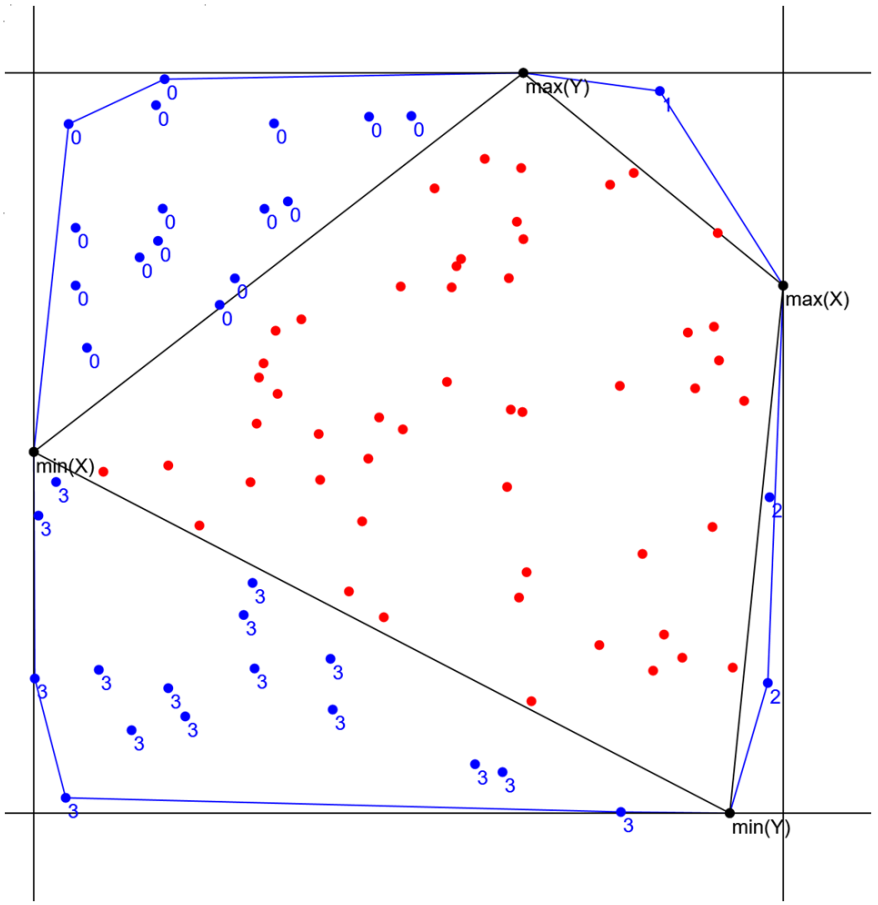


Figure 3. The Cutoff4 heuristic example.

the convex hull, are highlighted with blue circles, with labels indicating at which stage they were selected (e.g. label "0" means that these points are located "above" the segment connecting the min (X) and max (Y) points).

Given a uniform random distribution of points in a square region, we can expect that the Cutoff4 heuristic will, on average, halve the number of points to construct a convex hull. In the above example, 39 points out of 90 (including edge points) were selected.

A more significant reduction of points can be achieved if in addition to the specified edge points with the characteristics min (X), max (X), min (Y) and max (Y), we add points with the characteristics min ($X+Y$), max ($X+Y$), min ($X-Y$), max ($X-Y$). These edge points with respect to the lines with angles 45˚ and −45˚ are also obviously included in the convex hull. In this case, the final number of edge points can vary from 3 to 8. We will denote this heuristic by Cutoff8.

Figure 4 shows an example implementation of the Cutoff8 heuristic for the same set of source points as in **Figure 3**. In this example, the number of edge points is 7 because it turned out that the same point has the characteristics min (Y) and max ($X-Y$). The number of cutoff points is 76, the number of remaining points for the second stage of the Jarvis algorithm is 14 (including 7 edge points), *i.e.* reduced by more than 6 times.

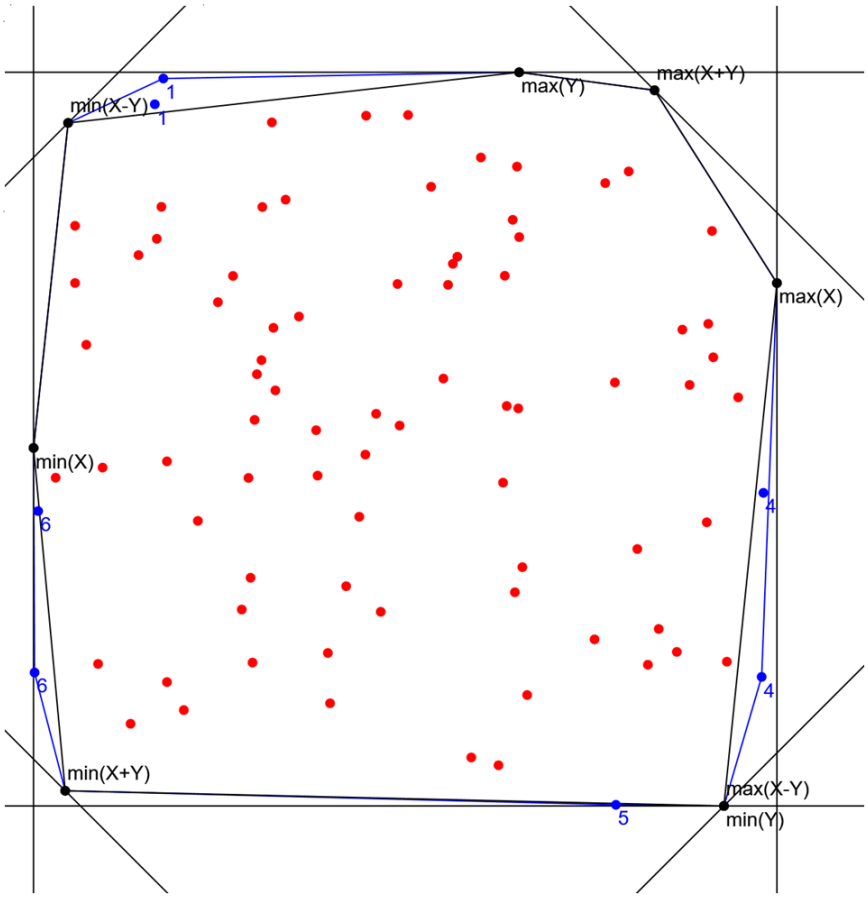


Figure 4. The Cutoff8 heuristic example.

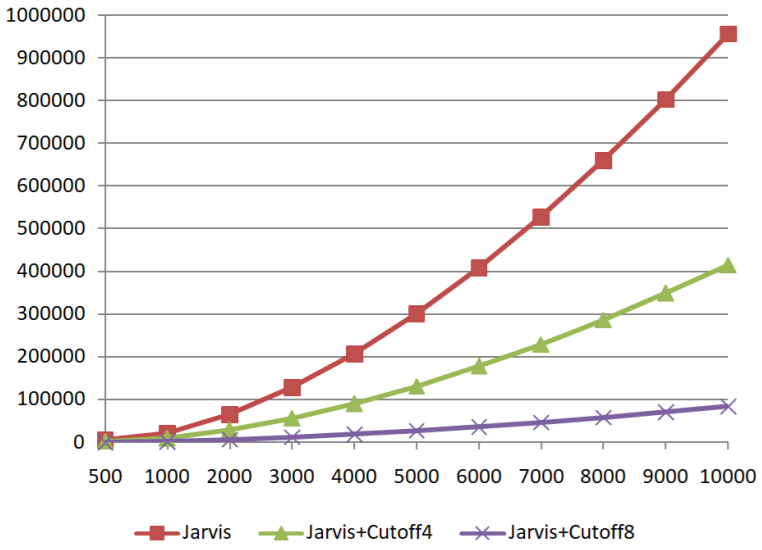


Figure 5. Average number of processed points for the original Jarvis algorithm and its versions with cutoffs (*OX* axis is the size of the initial set of points).

Figure 5 summarizes the results of different algorithms for constructing a family of convex contours for different numbers of initial points. The original Jarvis algorithm and its two modifications, supplemented with Cutoff4 and Cu-

toff8 heuristics, were used. For each number of points, 20 sets generated randomly in a square with side 1 were processed. The average number of points used in the main stage of the Jarvis algorithm (*i.e.* traversal) for all contours constructed is indicated. The average running times of these versions (in milliseconds) are summarized in **Figure 6**.

Because of the additional time spent on finding the edge points and on performing the cutoff actions, the acceleration of the algorithms with cutoff heuristics is not as large as one would expect based on the analysis of **Figure 5**, but it is significant enough to make it worthwhile to apply these modifications of the basic algorithm for a large number of points.

3. Merging Two Contours into a Single Path and Handling Special Cases

After finding a set of convex contours, we need to combine these contours into a single closed path. To do this, it is sufficient to select segments $[a_1, b_1]$ and $[a_2, b_2]$ of two neighboring contours and replace these segments with two new segments connecting the corresponding ends of the removed segments ($[a_1, a_2]$ and $[b_1, b_2]$ or $[a_1, b_2]$ and $[b_1, a_2]$). The choice of segments $[a_1, b_1]$ and $[a_2, b_2]$ should ensure the "minimality" of the obtained closed path. The simplest variant of such a choice, which we will call Simple, is the variant in which the added segments have total minimum length, *i.e.* the expressions dist (a_1, a_2) + dist (b_1, b_2) or dist (a_1, b_2) + dist (b_1, a_2) are minimal. As an example, **Figure 7(a)** shows a set of two contours (the points have coordinates between 0 and 1, as usual) and **Figure 7(b)** shows the result of their merging using the Simple algorithm (the segments of the contours connection are represented by lines of greater thickness). The total length of the resulting path is 3.05.

However, this method does not take into account the lengths of the contour

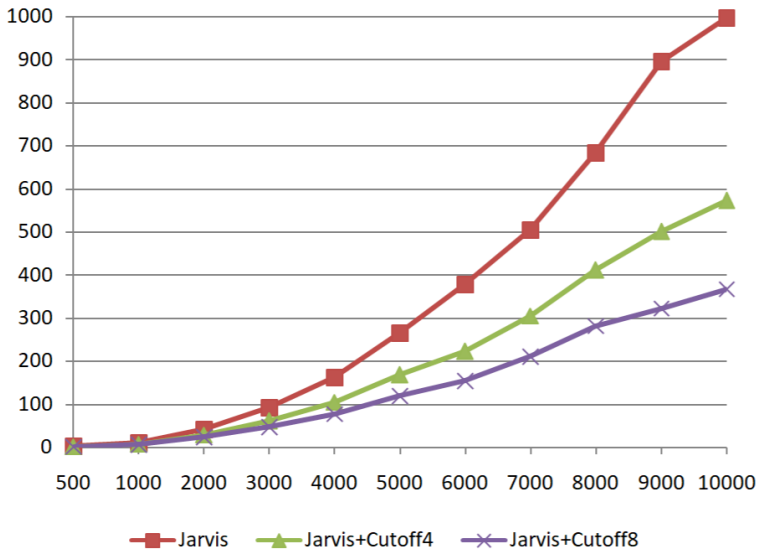


Figure 6. Average running time (in milliseconds) for the original Jarvis algorithm and its versions with cutoffs (*OX* axis is the size of the initial set of points).

segments to be removed. Therefore, a more optimal variant is the one that minimizes the expressions dist (a_1, a_2) + dist (b_1, b_2) − dist (a_1, b_1) − dist (a_2, b_2) or dist (a_1, b_2) + dist (b_1, a_2) − dist (a_1, b_1) − dist (a_2, b_2), in which the lengths of the removed segments are included with "+" sign, and the lengths of the added segments are included with "−" sign. The result of such a variant of the algorithm, which we will call AddSub, is shown in **Figure 7(c)**. In this case, the total path length is 2.77.

When programmatically implementing the algorithms of contours merging, it is convenient to use the contours descriptions in the form of a cyclic chain of points included in each contour. In the example shown in **Figure 7**, the outer contour is described by the chain (1, 4, 3, 2, 1) and the inner contour by the chain (6, 7, 5, 6). Note that exactly such chains are constructed in the Jarvis algorithm if the point with minimum abscissa min (X) is chosen as the initial point of the contour. In the Simple algorithm, link (2, 1) is removed from the first chain, link (5, 6) is removed from the second chain, and links (1, 6) and (2, 5) are added to the resulting path. In the AddSub algorithm, link (1, 4) is removed from the first chain, link (6, 7) is removed from the second chain, and links (1, 6) and (4, 7) are added to the resulting path. All other links of the original chains are retained; the result, for example, for the AddSub algorithm is the following change in the set of links (deleted links are underlined):

$$(\underline{1, 4}, 3, 2, 1), (\underline{6, 7}, 5, 6) \rightarrow (4, 3, 2, 1), (7, 5, 6) + (1, 6), (4, 7)$$

The same actions are required in the two special cases where the inner "contour" consists of one or two points (it is clear that for any set of nested convex contours there can be at most one such special contour). If we use representations of contours as cyclic chains, then the Simple and AddSub algorithms described above do not require any modifications. **Figure 8** and **Figure 9** illustrate these special cases. Note that in the case shown in **Figure 9**, the Simple and AddSub algorithms lead to the same result.

In the first case, we have cyclic chains (1, 4, 3, 2, 1) and (5, 5), while in the second case, we have (1, 4, 3, 2, 1) and (6, 5, 6). In all special case, we still remove two links from the original chains and add two links. For example, in the Simple algorithm, for the first case, we remove links (2, 1) and (5, 5) and add links (2, 5) and (1, 5):

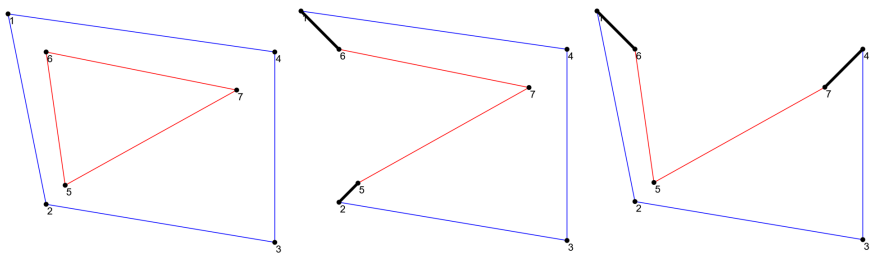


Figure 7. The original contours (a) and their merging by Simple (b) and AddSub (c) algorithms.

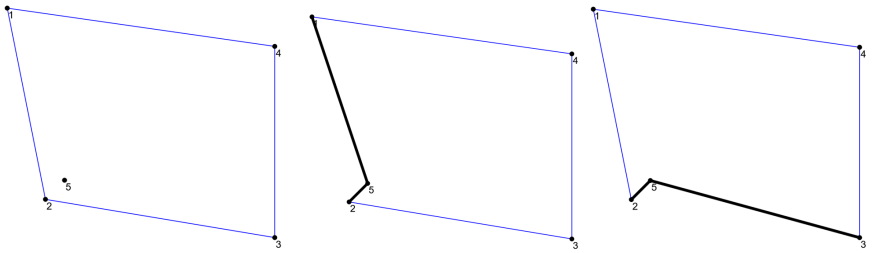


Figure 8. Case of degenerate single point inner contour: the original contours (a) and their merging by the Simple (b) and AddSub (c) algorithms.

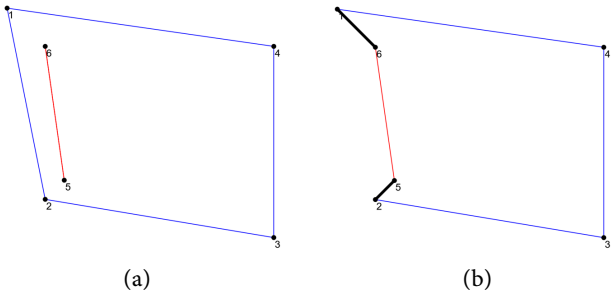


Figure 9. Case of a degenerate two-point inner contour: the original contours (a) and their merging by the Simple and AddSub (b) algorithms.

$$(1, 4, 3, \underline{2, 1}), (\underline{5, 5}) \rightarrow (1, 4, 3, 2), () + (1, 5), (2, 5)$$

In both algorithms for the second case, links (2, 1) and (6, 5) are removed and links (1, 6) and (2, 5) are added. Note that we retain the link (5, 6) of the second chain, as in the general case:

$$(1, 4, 3, \underline{2, 1}), (\underline{6, 5}, 6) \rightarrow (1, 4, 3, 2), (5, 6) + (1, 6), (4, 7)$$

Thus, the algorithm for merging two contours does not need to provide any additional handling of special cases.

4. Merging Several Contours into a Single Path: Algorithm Variants and Comparison of Their Efficiency

If there are more than two contours, then the merging process described above can be performed, for example, for each pair of neighboring contours, starting from the pair of outermost contours to the pair of innermost contours. When processing each subsequent pair of contours, the segment of the outer contour removed in the previous step must be excluded from consideration.

Figure 10 shows the image of a set of contours taken from **Figure 1** and the result of applying the AddSub algorithm to this set in the direction from the outer contours to the inner contours (we denote this variant by the Up algorithm).

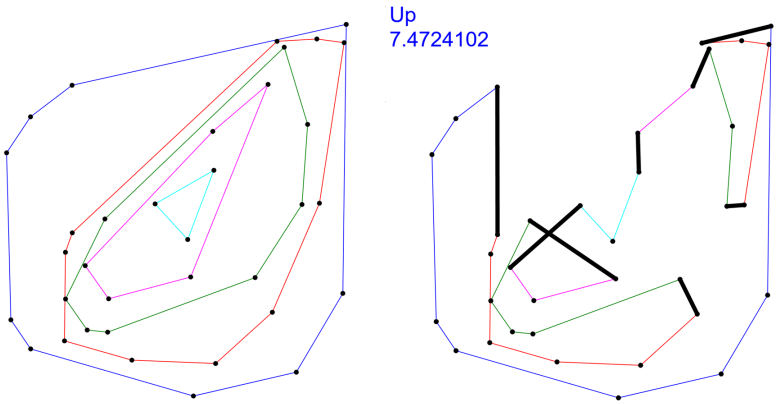
Note, however, that the choice of connecting segments reduces the number of possible choices for the next pair of contours. This may result in the loss of the optimal connection for the next pair of contours. Therefore, we can consider two

more modifications of the contour algorithm: in one of them (the Down modification) connections are built in the reverse direction (from inner to outer contour), and in the other (the "greedy" Mid modification) at each step all pairs of neighboring contours that have not been joined yet are analyzed and the contours of the pair for which the value of the AddSub expression described above is minimal are merged.

Examples of application of the Down and Mid modifications are shown in **Figure 11**. For this set, all three modifications give different results, and the Mid algorithm gives the best result.

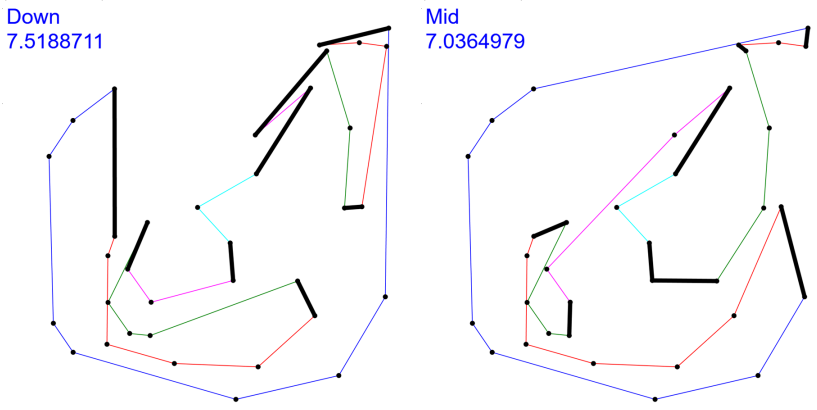
A numerical study of the above described Up, Down and Mid modifications of the contour algorithm was carried out. We use 20 randomly generated sets of 90 points located in a square with side 1. The initial parts of these sets consisting of 4, 6, 8, 10, 12, 15, 20, 25, 30, 35, 40, 50, 60, 70, 80 and 90 points were analyzed. **Figure 10** and **Figure 11** show an example of 35 points for one of the test sets.

The value



(a) (b)

Figure 10. Merging a set of contours (a) into a single path (b) by the Up algorithm.



(a) (b)

Figure 11. Merging a set of contours into a single path by the Down (a) and Mid (b) algorithms.

$$D_i = 100\ (S_i - S_0)/S_0 \tag{1}$$

was used to characterize the efficiency of the contour algorithm modifications, where S_0 is equal to the length of the final path obtained with the Simple algorithm (by enumerating the contours from outer to inner), and S_i is the length of the path obtained by one of the three modifications (Up, Down, Mid) of the Add-Sub algorithm. Thus, this value shows by how many percent the result has improved (*i.e. decreased*) compared to the Simple algorithm. The average values of D_i for the 20 test sets are summarized in **Figure 12**.

Thus, on average, the Mid algorithm gives the best results and Up algorithm comes next in terms of efficiency. However, the differences in the efficiency of these algorithms are negligible (of 1% - 2%).

5. Comparison of the Contour Algorithm with Other Approximate Algorithms for TSP Solving

It has been noted above that for the contour algorithm, results close to optimal cannot be expected. For a more accurate assessment of the applicability of this algorithm, we compared it with two known and quite efficient approximate algorithms for solving the traveling salesman problem. The first one is based on a variant of the method of branches and bounds [3] [4], and the second one uses the simulated annealing algorithm [5]. Both of these algorithms are real-time algorithms and provide the current pseudo-optimal solution at any point in their execution.

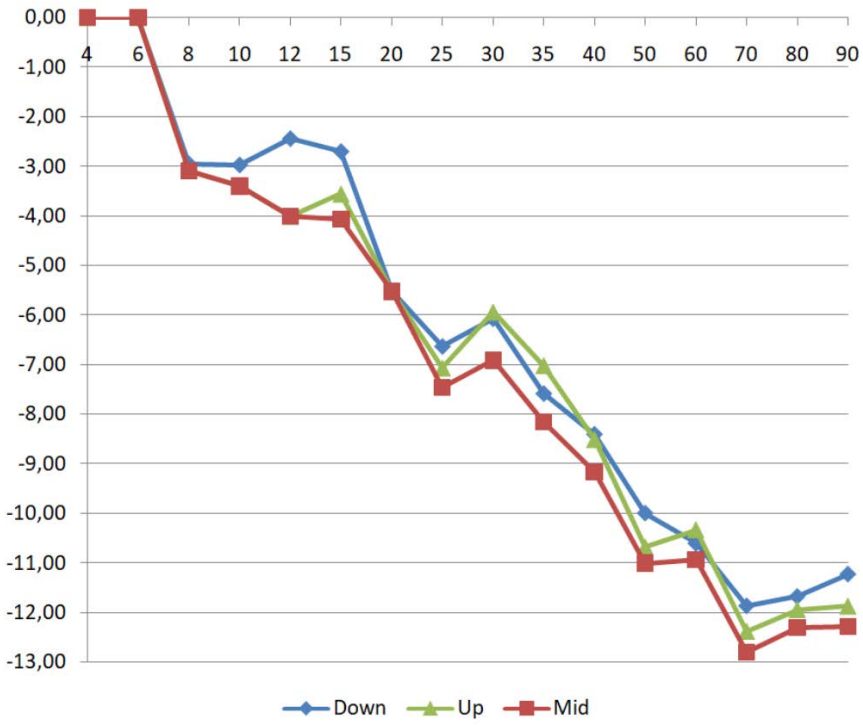


Figure 12. Improvement (in percent) of the results of Down, Up, Mid algorithms compared to Simple algorithm (*OX* axis is the size of the initial set of points).

Each of these algorithms was applied to the same test point sets as the contour algorithm. The Branch-bound algorithm was constrained to have a maximum number of branches of 100,000. A software implementation of the branch-and-bound method by one of the authors of this paper (B. F. Melnikov) was used.

Simulated annealing is a stochastic minimization method based on random wandering through space at successively lower temperatures, where the probability of performing a step is determined by the Boltzmann distribution [5]. For the simulated annealing algorithm, we used the function gsl_siman_solve of the GNU Scientific Library (GSL) with the following parameters: initial temperature T_INITIAL = 5000, final temperature T_MIN = 5.0e−6, Boltzmann constant K_B = 1.0, temperature reduction factor MU_T = 1.03.

Figure 13 shows the results of these algorithms for the set of 35 points previously considered for the contour algorithm (see **Figure 10** and **Figure 11**).

The Branch-bound algorithm, which showed the best results, was used as the base algorithm. Among the versions of the contour algorithms, the Mid algorithm was selected. To determine the relative efficiency of the Anneal and Mid algorithms compared to the Branch-bound algorithm, the previously described characteristic D_i (1) was used, for which in this case the final path length obtained by the Branch-bound algorithm was used as S_0 and the path length obtained by the Anneal and Mid algorithms was used as S_i. Here, this characteristic is positive and shows by how many percent the results *deteriorated* compared to the Branch-bound algorithm. The average values of D_i for the 20 test sets are summarized in **Figure 14**.

Figure 15 summarizes the minimum path values found by Branch-bound, Anneal and Mid algorithms for the first test set.

Thus, although the Branch-bound and Anneal algorithms give better results, the deterioration of the Mid algorithm results does not exceed 90% even for 90 points.

Large sets of points (500, 1000, ..., 10,000) were also analyzed. In this case, the branch and bound method was able to process only sets of size 500 and 1000, as

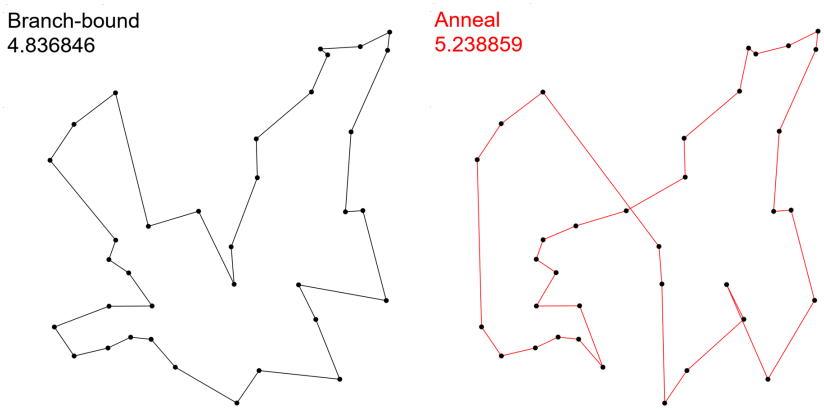


Figure 13. Paths obtained by Branch-bound (a) and Anneal (b) algorithms.

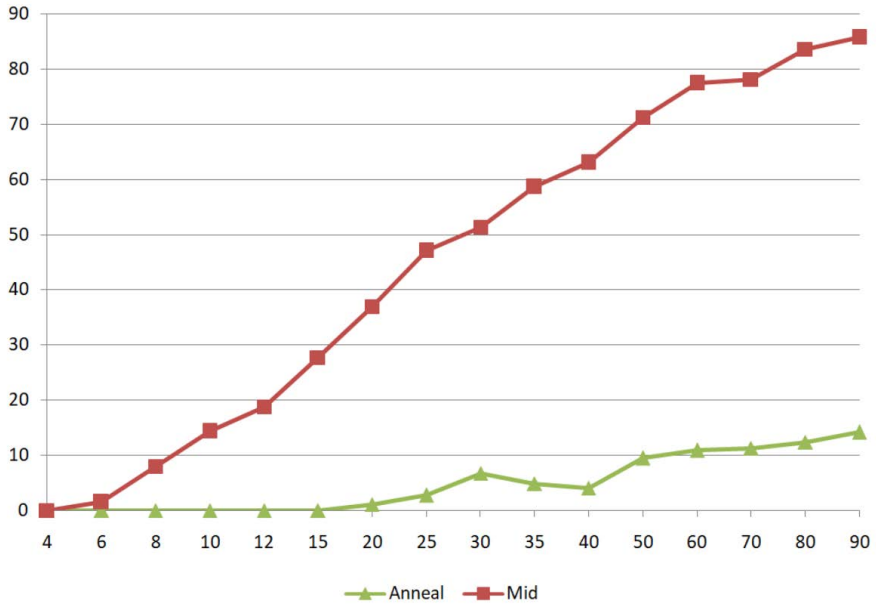


Figure 14. Deterioration (in percent) of Anneal and Mid algorithm results compared to the Branch-bound algorithm (*OX* axis is the size of the initial set of points).

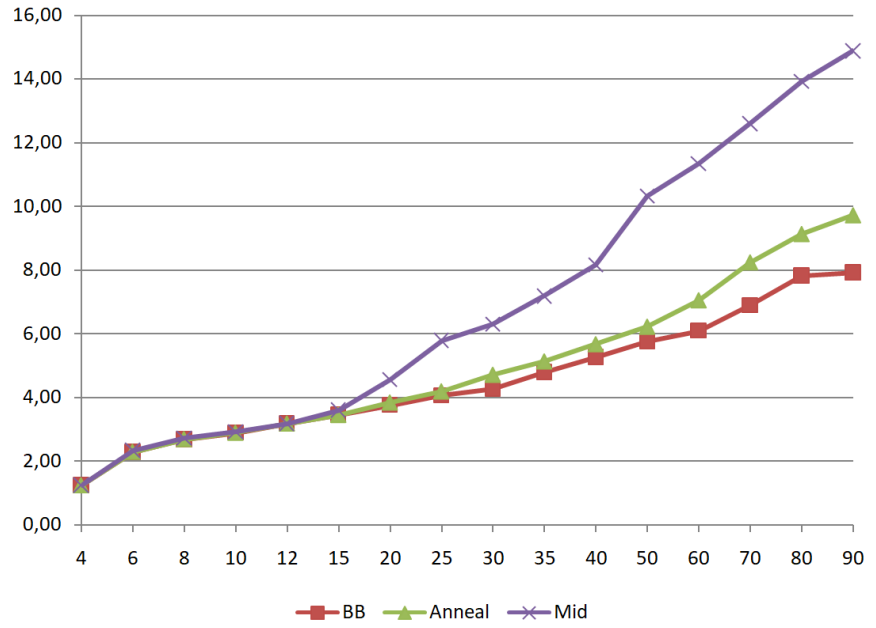


Figure 15. Results of Branch-bound, Anneal and Mid algorithms for the first test set (*OX* axis is the size of the initial set of points).

the available memory was insufficient for larger sets. It should also be noted that the running time of the simulated annealing method varied from 98 seconds (for 500 points) to 45 minutes (for 10,000 points), while the Mid algorithm ran less than 1 second for all sets.

Figure 16 summarizes the results for the first larger test set. The number of contours varied from 31 (for 500 points) to 225 (for 10,000 points).

The numerical results of the algorithms for large size point sets show that

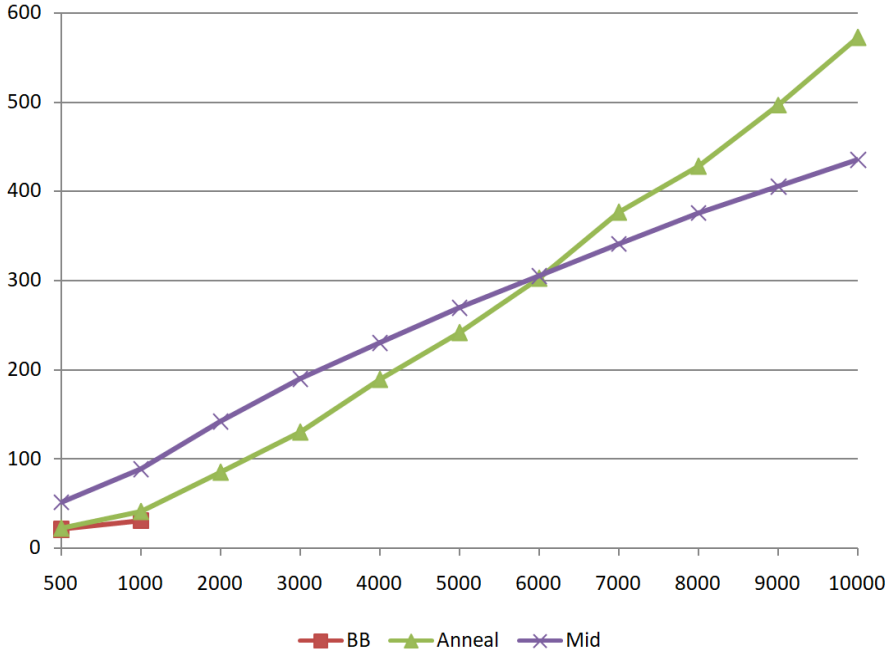


Figure 16. Results of Branch-bound, Anneal and Mid algorithms for the first test set of large size (*OX* axis is the size of the initial set of points).

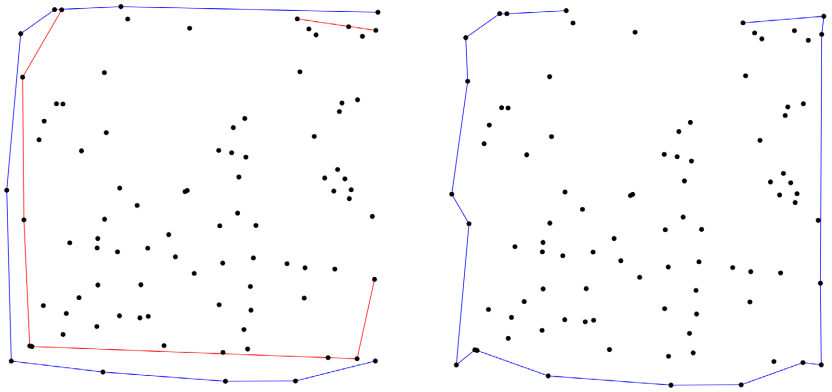


Figure 17. Segments of the two outer contours remaining after executing the Up algorithm (a) and a variant of the more efficient merging two outer contours (b) for a set of 90 points.

starting with point sets of a certain size, the Mid algorithm performs better than other approximate algorithms and also runs much faster.

6. Conclusions

The paper considers various aspects of the implementation of the contour algorithm related to both the first stage (construction of a set of convex contours) and the second stage (merging contours into a closed path). Numerical comparison with other known algorithms for approximate solution of the traveling salesman problem shows that for a small number of points, the deterioration of the

results obtained using the described variants of the contour algorithm does not exceed 90%. For a large number of points (starting from 6000), the contour algorithm allows to obtain better results, and for a very short running time (about 1 s).

The main problem with the considered variants of the contour algorithm is that neighboring contours are bypassed completely except for one or two removed segments (**Figure 17(a)**). It can be expected that the inclusion of additional heuristics in the algorithm to provide a more efficient merging of neighboring contours (**Figure 17(b)**) will significantly improve its performance. The development and analysis of such heuristics is the subject of future research.

Founding

This work is supported by a grant from the research program of Chinese universities "Higher Education Stability Support Program" (Section "Shenzhen 2022—Science, Technology and Innovation Commission of Shenzhen Municipality").

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Garey, M. and Johnson, D. (1979) Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., San Francisco.

[2] Applegate, D.L., Bixby, R.E., Chvátal, V. and Cook, W.J. (2002) The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton.

[3] Goodman, S. and Hedetniemi, S. (1977) Introduction to the Design and Analysis of Algorithms. McGraw-Hill, New York.

[4] Melnikov, B. and Melnikova, E. (2022) On the Classical Version of the Branch and Bound Method. *Computer Tools in Education*, **2**, 41-58.

[5] Aarts, E.H.L. and Korst, J. (1989) Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. Wiley, Chichester.