Octopus Algorithm as a New Support in Solving TSP

Marek Sosnicki, Iwona Pozniak-Koszalka, Leszek Koszalka, Andrzej Kasprzak
Dept. of Systems and Computer Networks
Wroclaw University of Science and Technology
Wroclaw, Poland
e-mail: {marek.sosnicki, iwona.pozniak-koszalka, leszek.koszalka, andrzej.kasprzak}@pwr.edu.pl

***Abstract*—In this paper, a designed and implemented algorithm, named Octopus is applied for solving the Travelling Salesman Problem (TSP). In general, the Octopus algorithm can be used as both a method of finding good solutions of the optimization problem and a way to get starting points for other meta-heuristic algorithms used in problem solving, for instance Tabu Search (TS). Octopus takes into account multiple solutions gathered by the meta-heuristic algorithms and combines them to obtain new ones. The results of simulation experiments show that Octopus may be considered as very promising.**

Keywords-algorithm; TSP; Tabu Search; experimentation system; simulation.

I. INTRODUCTION

TSP is one of the most popular optimization problems. The problem consists in finding a path between points (e.g., cities on the map) with known location. The path (route) has to allow for visiting all points as fast as possible or using the shortest possible route. In practice, when the best route between dozens or hundreds of points should be found then one can realize that it is a real NP-hard problem (nondeterministic polynomial) [1].

Many authors have tried to solve this problem using different approaches, e.g., Branch and Bound [2], but such methods can provide solution in reasonable time only to small instances of the problem, namely, a few locations, only. Another way to find the solution is using heuristic algorithms. These algorithms can provide solutions, which are close to optimal ones but the process of finding them is much quicker than for exact methods. There are many types of heuristics which can be used. It is worth to mention such algorithms as Parallel Evolutionary [3], Artificial Bee Colony (ABC) [4], Ant Colony Optimization (ACO) [5][6], Tabu Search (TS) [7]-[9].

We are interested in algorithms which have some starting solutions and then they try to construct improved solutions by slightly changing them and checking, i.e. optimizing them. Such a process is then repeated multiple times until getting the best or satisfying solution. Local search algorithms are great in finding good solutions in very short time, but they have one disadvantage. They tend to get stuck in local minimum. It usually happens when we cannot find any way to change our current solution so that it gets better in next iterations.

The Octopus algorithm proposed in this paper deals with the problem of getting stuck in local minimum areas with local search algorithms. The proposed approach allows combining multiple different algorithms, anticipating that they can give better results when used together while working in parallel environments [10]. The algorithm requires activities on three stages: (*i*) taking many good solutions from local search algorithms (starting from different random solutions); (*ii*) combining these solutions and generating new starting points, which keep some information about good solutions; (*iii*) running algorithms again from the new starting points.

The rest of the paper is organized as follows. In Section II, the mathematical model of the considered TSP problem is formulated, and we provide the justification why TSP was chosen for investigation concerning the proposed Octopus algorithm. In Section III, the implementation of Tabu Search which was taken as meta-heuristic algorithm for testing a new algorithm is briefly described, as well as the core of the paper – the Octopus algorithm is presented in detail. The presentation of research has been contained in Section IV. Investigation concentrates on properties of the proposed algorithm, and its advantages in comparison with known algorithms to solving the considered TSP problem. The conclusion and plans for the future work to improving Octopus algorithm appear in the last Section V.

II. PROBLEM FORMULATION

The considered TSP problem can be formulated as follows.

Given:

- A graph G = (V; E), where V = {1, 2, …, n} is a vertex set of points and E = {{i, j}: i ≠ j, i, j ϵ V} is an edges set (e.g., paths between cities).
- A nonnegative cost (distance) matrix C = $[c_{i,\ j}]$ defined on E.

To find

- A route π = (π(1), π(2),…,π(n)) ϵ Π, which is represented as permutation of vertices [3] that passes through each point exactly once, and returns to the starting point on a G.

Such that

- The optimal solution (π*) minimizes the cost (fitness) of the route defined as the arithmetic sum of all paths between points, which belong to π.

 ISBN: 978-1-61208-696-5

In this paper, only the symmetric version of TSP is considered as it is a more common usage, but the proposed algorithm can also work for the asymmetric version.

There are three main reasons why the above problem was selected for testing the performance of Octopus algorithm.

- Every permutation of nodes in this problem is a feasible solution. That allows us to mix solutions easier. In many optimization problems it is hard to even find a random permutation, as most of possible permutations are not feasible.
- The local search algorithms work very well and fast in this problem. It is due to the fact that it is easy to find a value of a fitness function for neighbor solutions.
- There are many local search metaheuristic algorithms invented for solving TSP, so the new algorithm may become helpful to combine them.

The proposed algorithm should also work for any other problem, which fulfills the first reason.

III. OCTOPUS ALGORITHM

In the first subsection, the collaborating algorithm Tabu Search (TS) is briefly presented. The next subsections describe three stages of the performance of the Octopus algorithm.

A. Tabu Search for Testing

For testing the Octopus algorithm, the implemented version of TS algorithm was taken. The pseudocode for TS [11] is presented in Figure 1.

```
⌂ produce an initial solution π;
π* ← π;
initiate tabu list T;
while termination criterion is not satisfied do
        get all neighbors N(s) according to π and T;
        find the best solution π^b, in N(s);
        π* ← π^b,;
        update T;
          if fit(π) ≤ fit(π*) then
          π* ← π,;
          end if
end while
return π*
```

Figure 1. Pseudocode of the implemented Tabu Search.

The neighborhood is generated as all possible swaps of two elements of permutation. The tabu list is saving two indices of permutation, which were swapped in the previous iteration. Tabu list size is fixed, e.g., could be set to 100. The termination criterion is taken as the number of iterations, which is a parameter of the algorithm. The initial solution is treated as another TS parameter.

B. The First Stage – Local Search

The first stage of the algorithm is gathering solutions. The exemplary metaheuristic TS algorithm is running N times starting from random permutations, with the same number of iterations in each run. Several performances of TS could run in separate threads as they do not need to exchange information between each other. To obtain satisfying results, the minimum of N is fixed as N=50, thus different solutions must be collected. It can be noticed that the obtained permutations are similar to each other in a way the distances to each other [12], are smaller than distances to permutations chosen at random.

C. The Second Stage – Combining Solutions

After Octopus gathered permutations $P=\{\pi_1, \pi_2,\ldots, \pi_N\}$, by performing in the previous stage, it makes changes in the introduced matrix called Order Matrix (OM). The matrix OM is created taking into account all permutations. An element of OM denoted as $a_{i,j}$ is equal to 1 if $\pi^{-1}(i) < \pi^{-1}(j)$, and is equal to 0, otherwise. Then, all matrices are combined together by summing them and creating the matrix S expressed by (1).

$$S = A_1 + A_2 + \ldots + A_N \quad (1)$$

D. The Third Stage – New Starting Points

To get the new start points, the procedure shown in Figure 2 is proposed.

```
⌂ produce an initial permutation
π ← {1};
while π ≠ n d
      ⌂ Repeated until permutation has all elements
      i* ← 0;
      v* ← -∞;
      for all i ϵ {2, 3,…, n} do
         if i ϵ π then
         continue;
         end if
         for all Possible insert positions p do
                ⌂ p = 0 is inserted between all elements,
                      ⌂ p = 1 is after first, etc.
            Calculate v = V (π, i, p);
            if  v > v* then
                  v* ← v;
                  p* ← p;
                  i* ← i;
            end if
         end for
      end for
      insert i* into π at p* position;
end while
return π
```

Figure 2. Pseudocode of the implemented Tabu Search.

The introduced formula to calculate the inserted value is expressed by (2).

$$V(\pi, i, p) = \Sigma\eta_{p,k} \left(S_{i,\eta(k)} - S_{\eta(k),i} \right) \quad (2)$$

where:

- The summation in (2) goes from $k=0$ to the absolute value of p;
- The coefficient $\eta_{p,k}$ introduced in (2) is equal to -1 if $p < k$ or is equal to +1, otherwise.

This formula is proposed to select such inserts which are most common among the permutations calculated in the first stage. That means, e. g., if node 6 is always after 5 in the obtained solutions, then putting 6 after 5 in permutation would have very high insert value.

After such new solution π is obtained, the OM of this solution is subtracted from S and the procedure is repeated to find the next solution (based on the new matrix S).

Unfortunately, the proposed approach allows us to obtain about $M=0.2*N$ reasonable solutions. When we try to generate more permutations than M, the produced permutations become more and more random, and do not have the desirable features that can be observed in the first permutations. After we obtain M new permutations, we run our local search algorithms on them to obtain new, better solutions. We can repeat these stages all over, but we must remember that the number of processed permutations may decrease by 80% with each third stage of Octopus.

IV. RESEARCH

The main goal of the research was to check whether the Octopus algorithm can give good solutions to the considered TSP, and to observe if the features of good solutions are remembered in the new permutations when using the proposed approach.

The simulation experiments have been made taking into account five instances from TSPLib., namely kroA100, kroB100, kroC100, kroA200, kroB200 [13]. The first three instances concern TSP with 100 nodes (points), and the next two instances concern TSP with 200 nodes (points).

A. Testing TS

Firstly, Tabu Search algorithm was tested for random permutations generated as start points, to show that the implementation can give promising results. TS was run with different number of iterations.

The results obtained for kro100A test instance are presented in Figure 3. It may be observed that the implemented TS algorithm produced good solutions for not a big number of iterations and that the first 200 iterations were crucial.

Table 1 shows the results obtained by TS with 1000 iterations in comparison to the best known solutions for the considered instances. Also, it can be seen that the solutions found by TS are worse only approximately 10 % in average than the best solutions listed in TSPLib. This can justify the fact that the algorithm works properly and can give satisfactory results.

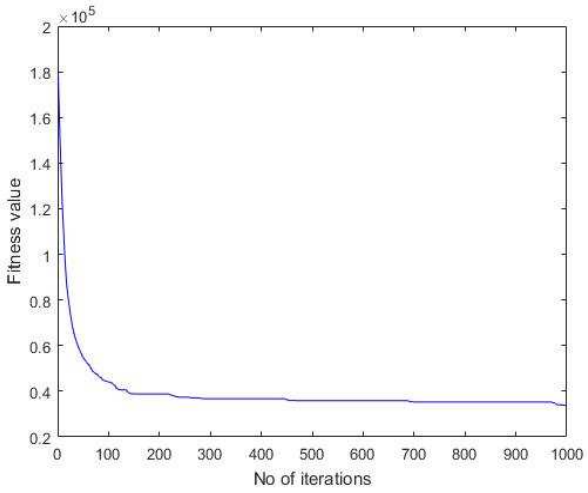

Figure 3. TS algorithm for kroA100 instance.

TABLE I. COST FUNCTION FOR SOLUTIONS OF TSP

Test instance	TS result	Best known
kroA100	23405	21282
kroB100	24240	22141
kroC100	22733	29749
kroA200	35456	29368
kroB200	35311	29437

B. Testing Octopus

Next, we checked if the new start points obtained when Octopus is utilized can ensure better solutions than these obtained when random start points are taken into account. A single simulation experiment consisted of six steps. Each experiment was conducted using following the procedure:

***Step* 1**. The number of 100 permutations was chosen at random.

***Step* 2.** TS was applied for each of them for 5000 iterations (in case of problems with 200 nodes) or 1000 iterations (in case of problems with 100 nodes).

***Step* 3**. Octopus algorithm was run the first time on the obtained solutions by TS and 20 new start points were created.

***Step* 4.** TS was run again using these 20 solutions (start points) for the same number of iterations.

***Step* 5**. The second run of Octopus algorithm was performed and finally four solutions were taken.

***Step* 6**. TS algorithm was performed once more using these four solutions.

 ISBN: 978-1-61208-696-5

In Figure 4, the exemplary results of solving TSP for instance kroB200 with two runs of Octopus are shown.

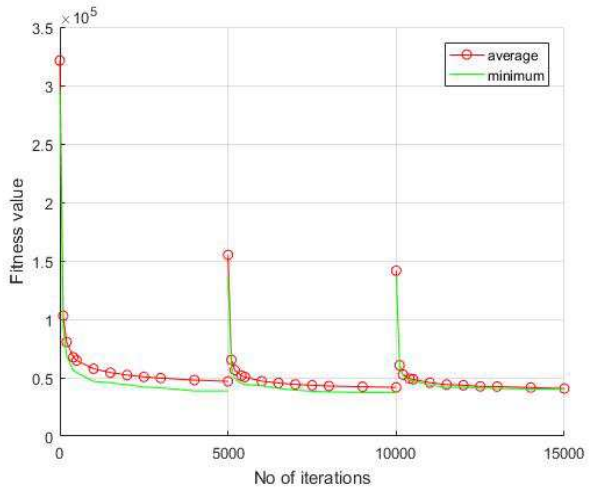

Figure 4. Octopus – runs with 5000 iterations for kroB200.

In Figure 5, the results for the same case are presented but with the first 100 iterations of TS removed in order to increase readability.

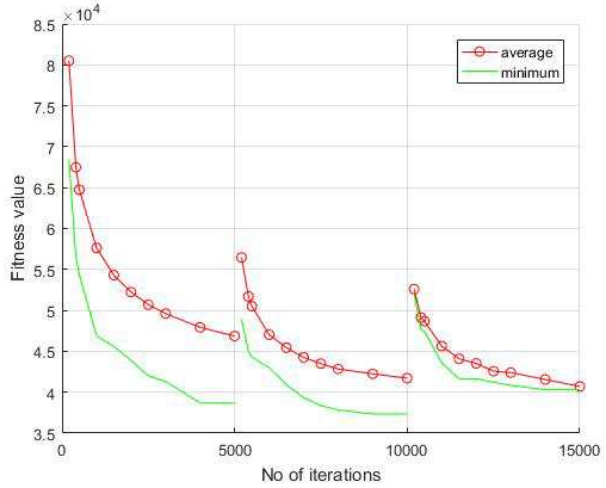

Figure 5. Octopus – runs with 5000 iterations (first 100 removed in each run) for kroB200.

C. Testing TS with Octopus

The aim of this complex experiment was the comparison of results obtained in solving TSP for the considered instances with a sequence of 3 runs:

***Run 0*:** Using the standard version of TS with start points generated randomly.

Run I. Using TS with the first run of Octopus (with new start points).

Run II. Using TS with the second run of Octopus (with additional start points).

Table II presents the results obtained for five instances. In columns show the average values of the fitness function (denoted as Avg) for the start points in successive runs.

TABLE II. AVERAGED FITNESS VALUES OF START POINTS

Test instance	Avg (0)	Avg (I)	Avg (II)
kroA100	170657	101347	86391
kroB100	170696	89900	78432
kroC100	169766	104592	89334
kroA200	340429	205334	178054
kroB200	334684	198752	171779

We can see that the first run of Octopus gave much better start points than random ones for all considered instances. We define Profit as the percentage decrease in the fitness function, as expressed by (3).

$$\text{Profit } (x+1) = \{[\text{Avg } (x) - \text{Avg } (x+1)] / \text{Avg } (x)\} * 100\ \%. \quad (3)$$

where x = 0, 1. In Table III, the values of the obtained Profit are specified. The Profit from the first run of Octopus was of more than 40 % and may be considered relatively large. It indicates that Octopus can be used to find new starting points based on some already known solutions. The second run of Octopus was not so effective, giving a modest increase in quality of 13.7 %, which is almost three times less than the first run of Octopus.

TABLE III. PROFIT GIVEN BY OCTOPUS

Test instance	Profit (I)	Profit (II)
kroA100	40.6 %	14.8 %
kroB100	47.1 %	12.8 %
kroC100	38.2 %	14.4 %
kroA200	39.7 %	13.1 %
kroB200	40.7 %	13.6 %
mean	41.3 %	13.7 %

Table IV presents the results obtained for all instances after Run (0) and Run (I) with Octopus. The notation Min (x) represents the best solution of TSP.

TABLE IV. SOLUTIONS MADE WITH OCTOPUS RUNS

Test Instance	Min (0)	Avg (0)	Min (I)	Avg (I)
kroA100	25690	31177	24538	28517
kroB100	26617	31780	26090	28724
kroC100	26829	30659	25472	30378
kroA200	38882	44300	37508	42185
kroB200	38499	43888	38232	41199

The analysis of the results presented in Table IV confirms that using Octopus can improve the process of finding the solution to TSP with the implemented TS.

 ISBN: 978-1-61208-696-5

V. CONCLUSION AND FUTURE WORK

Based on the results of experiments, we can say that the proposed Octopus algorithm seems to be promising. The basic idea of this algorithm is to combine solutions in order to create new ones, by keeping the good features of the old ones. The new start points that are created by Octopus possess some features of the old permutations. Using Octopus gives a chance to avoid being stuck in local minimum areas.

However, the proposed approach is not perfect. The drawback is that, with each step of the algorithm, we reduce the number of processed solutions by 80%. Moreover, we are starting from random points, and we cannot be sure if after using Octopus algorithm we would not try to check some permutations multiple times, which would be a waste of processing power.

In our further work, we plan:

- To apply Octopus algorithm for solving optimization problems other than TSP.
- To use Octopus in collaboration with other local search algorithms.
- To improve getting more reasonable start points from a single run.
- To consider possibility of implementing Octopus on GPU (Graphics Processing Unit) threads, as well as using MPI (Message Passing Library).
- To implement an experimentation system along with the rules described in [14].

ACKNOWLEDGMENT

This work was supported by the statutory funds of the Department of Systems and Computer Networks under grant no. 0401/0132/18, Faculty of Electronics, Wroclaw University of Science and Technology, Wroclaw, Poland.

REFERENCES

[1] K. Ilavarasi and J. K. Suresh, " Variants of Traveling Salesman Problem: A Survey," Proc. of International Conference on Information, Communication and Embedded Systems (ICICES), February 2014, pp. 1-7, IEEE, doi:10.1109/ICICES 2014.70338550.

[2] R. Grymin and S. Jagiello, "Fast Branch and Bound Algorithm for the Traveling Salesman Problem," Proc. of IFIP conference on Computer Information Systems and Industrial Management, 2016, pp. 206-217.

[3] W. Bozejko and M. Wodecki, "Parallel Evolutionary Algorithm for the Traveling Salesman Problem", Journal of Numerical Analysis, Industrial and Applied Mathematics, 2 (3-4), 2007, pp. 129-137.

[4] N. Pathak and S. P. Tiwasi, "Traveling Salesman Problem Using Bee Colony with SPV," International Journal of Soft Computing and Engineering (IJSCE), vol. 2, July 2012, pp. 410-414.

[5] K. Baranowski, L. Koszalka, I. Pozniak-Koszalka, and A. Kasprzak, "Ant Colony Optimization Algorithm for Solving the Provider – Modified Travelling Salesman Problem" Proc. of ACIIDS conference, April 2014, Springer, Lectures Notes in Computer Science, pp. 493-502.

[6] P. Jarecki, P. Kopec, I. Pozniak-Koszalka, L. Koszalka, and A. Kasprzak, "Comparison of Algorithms for Finding Best Route in An Area With Obstacles," Proc. of International Conference on Systems Engineering (ICSEng), Las Vegas, USA, August 2017, pp. 163-168.

[7] Y. He, Y. Qiu, G. Liu, and K. Lei, "A Parallel Adaptive Tabu Search Approach for Traveling Salesman Problem," Proc. of IEEE Internat. Conference on Natural Language Processing and Knowledge Engineering, November 2005, pp. 796-801.

[8] Y-F. Lim, P-Y. Hong, R. Raml, and R. Khalid, "An Improved Tabu Search for Solving Symmetric Traveling Salesman Problems," Proc. of IEEE Colloquium on Humanistic Science and Engineering (CHUSER), December 2011, pp. 851-854.

[9] E. Osaba and F. Daz, " Comparison of Memetic Algorithm and Tabu Search Algorithm for the Traveling Salesman Problem," Proc. of Federated Conference on Computer Science and Information Systems. (FedCSIS), September 2012, pp. 131-136.

[10] W. Yen and D. McLean, "Combining Heuristics for Optimizing A Neural Net Solution to The Traveling Salesman Problem," Proc. of International Joint Conference on Neural Networks (IJCNN), June 1990, pp. 259-264.

[11] Y-W. Zhong, C. Wu, L-S. Li, and Z. Y. Ning, "The Study of Neighborhood Structures of Tabu Search Algorithm for Traveling Salesman Problem," Proc. to 4th International Conference on Natural Computation (ICNC), October 2008, p. 491.

[12] A. B. Rathod, "A Comparative Study on Distance Measuring Approaches for Permutation Representations," Proc. of IEEE International Conference on Advances in Electronics Communication and Computer Technology (ICAECCT), December 2016, pp. 251-256.

[13] TSPLibrary [Online]. Available from: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95 [retrieved: December 2018].

[14] M. Hudziak, I. Pozniak-Koszalka, L Koszalka, and A. Kasprzak, "Multiagent Pathfinding in the Crowded Environment with Obstacles," Journal of Intelligent and Fuzzy Systems 32(2), 2017, pp. 1561-1573.

 ISBN: 978-1-61208-696-5