Open Access Library Journal
2021, Volume 8, e7520
ISSN Online: 2333-9721
ISSN Print: 2333-9705

Repetitive Nearest Neighbor Based Simulated Annealing Search Optimization Algorithm for Traveling Salesman Problem

Md. Azizur Rahman[1*], Hasan Parvez[2]

[1]Mathematics Discipline, Science, Engineering and Technology School, Khulna University, Khulna, Bangladesh
[2]Navy Anchorage School and College Khulna, Sailors' Colony, Goalkhali, GPO, Khulna, Bangladesh
Email: *mdazizur@math.ku.ac.bd

How to cite this paper: Rahman, Md.A. and Parvez, H. (2021) Repetitive Nearest Neighbor Based Simulated Annealing Search Optimization Algorithm for Traveling Salesman Problem. *Open Access Library Journal*, **8**: e7520.
https://doi.org/10.4236/oalib.1107520

Received: May 13, 2021
Accepted: May 30, 2021
Published: June 2, 2021

Abstract

The traveling salesman problem (TSP) is the most popular and most studied non-deterministic polynomial (NP) hard problem that has been used in various fields of science and technology. Due to the NP-hard nature, it is very difficult to solve this problem effectively and efficiently. For this reason, diverse appropriate optimization algorithms have been designed and developed in the last few decades. Among these algorithms, heuristic algorithms are much more suitable to tackle with this complex problem. In this paper, we propose a hybrid heuristic algorithm to solve the symmetric TSP problem by combining the search mechanism of repetitive nearest neighbor (RNN) heuristic and simulated annealing (SA) heuristic algorithms. In fact, a set of better routes are generated step by step by the RNN algorithm and these routes are improved through the iterative improvement process of the SA algorithm. The proposed algorithm is tested on a set of benchmark symmetric TSP datasets and compared with the basic RNN and SA algorithms as well as some other hybrid algorithms existing in the literature. It is demonstrated by the experimental results that the proposed algorithm is more effective than both the basic RNN and SA algorithms, and the obtained optimum results are in good agreement with the corresponding best-known optimum results. In addition, the proposed algorithm outperforms some other hybrid algorithms in terms of solution quality.

Subject Areas

Intelligent Optimization Algorithm, Combinatorial Optimization

Keywords

Combinatorial Optimization, Traveling Salesman Problem, Repetitive

*Corresponding author.

Nearest Neighbor Algorithm, Simulated Annealing Algorithm, Neighborhood Structure, Hybrid Algorithm

1. Introduction

The traveling salesman problem (TSP) is a well-known combinatorial optimization problem that has been extensively studied in various fields of science and technology such as mathematics, artificial intelligence, physics, operations research, and biology. It is the problem of finding the possible shortest route among a list of cities, where each city is included once and only once and finally returns to the starting city. It is believed that the history of the TSP problem was discovered in 1920 in Vienna [1]. However, a formal description of the TSP problem was formulated by Dantzig *et al.* in 1954 [2]. Since then it has been used in modeling diverse real-world problems, such as designing hardware devices, microchips, and radio electronic devices, data association, data transmission in computer networks, DNA sequencing, vehicle routing, job scheduling, clustering of data arrays, image processing and pattern recognition, analysis of the structure of crystals, transportation, logistics, supply chain management, etc. [3] [4]. The TSP problem is easy to understand but often very difficult to solve as it contains all features of the combinatorial optimization problem. In fact, it has been proven to be a non-deterministic polynomial (NP) hard problem [5]. By NP-hard, we mean those problems for which no polynomial time algorithm exists to effectively solve them. Indeed, the executive time of any existing algorithm for solving the TSP problem is increased super-polynomially (or, perhaps exponentially) with the number of cities [4]. Thus, the study on improving the solution algorithm of the TSP problem has important theoretical, engineering, and practical significance.

In graph theory, the TSP problem can be illustrated by a complete directed graph $G=(V,E)$, where V represents the set of cities also called nodes or vertices and E denotes the set of edges also called the path between each pair of distinct vertices. A distance (cost) matrix $D=d_{ij}$ is associated with each edge $e_{ij}\in E$ also called the edge weight. Depending on the distance (cost) matrix D, the TSP problem can be categorized as symmetric or asymmetric. The graph G is called a symmetric TSP if all the edges of G are symmetrical edges, *i.e.*, $d_{ij}=d_{ji}$ $\forall e_{ij}\in E$. On the other hand, G is called a asymmetric TSP if there exists at least one edge $e_{ij}\in E$ for which $d_{ij}\neq d_{ji}$. In this study, we consider the asymmetric TSP problem. Thus, the objective function Z of the TSP problem can be formulated as follows [3]:

$$Z=\min\sum_i\sum_j d_{ij}x_{ij};\ \ x_{ij}=\begin{cases}1;\text{ if the edge } e_{ij} \text{ is in the route}\\ 0;\text{ otherwise}\end{cases} \tag{1}$$

with respect to the following constrains:

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,2,3,\cdots,n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,2,3,\cdots,n \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad i, j = 1,2,3,\cdots,n \tag{4}$$

$$\sum_{i,j \in S}^{n} x_{ij} \le |S| - 1, \quad 2 \le |S| \le n-2 \tag{5}$$

In the above model, Equation (1) is the total distance (cost) that needs to be minimized and Equations (2) - (5) are the constraints' condition of the model. Equation (2) ensures that each location *j* is occupied by only one city, whereas Equation (3) guarantees that each city *i* is assigned one exact position. Equation (4) refers to the integrality constraints of variables zero-one $x_{ij}\left(x_{ij} \succ 0\right)$. In contrast, Equation (5) assures that each city in the final route will be visited once and that no sub-routes will be formed.

Due to the applicability and complexity, various researchers have been designed and developed different optimization algorithms in the last few decades to deal with the TSP problem. Among these algorithms, heuristic algorithms are the most successful and widely used search mechanism for solving the TSP problem. Heuristic algorithms, however, offer a satisfactory solution but are often meet with the problem of premature converge. As a result, the search process easily falls into the trapped of local optimum condition and is unable to jump the solution into another promising search space. For this reason, many researchers turned into developing hybrid optimization algorithms through the integration of the superiority of two or more heuristic optimization algorithms. The aim of this paper is to design and implement a hybrid optimization algorithm called RNN-SA for solving symmetric TSP problem, which uses the search process of RNN algorithm and SA algorithm. The proposed RNN-SA algorithm performs the search procedure by two stages. It first generates a set of feasible routes step by step based on the procedure of RNN. Then, these routes are used as the initial solutions of the SA algorithm and are improved iteratively in an effective iterative improvement process. The proposed algorithm is implemented on a collection of benchmark TSP datasets. The proposed RNN-SA gets better results than both the RNN and SA algorithms and performs better than some other hybrid optimization algorithms.

The rest of this paper is organized as follows. In Section 2, we review some related hybrid algorithms for solving the TSP problem. In Section 3, we briefly introduce the optimization approaches under consideration in this study that make up the proposed hybrid algorithm. In Section 4, we discuss the proposed hybrid RNN-SA algorithm in detail. In Section 5, we present experimental results, result analysis, and performance comparisons. The conclusion of the paper is summarized in Section 6.

2. Related Work

In this section, we review some recently published hybrid optimization proce-

dures that use different strategies to develop sophisticated solution methods for solving the TSP problem. Geng *et al.* proposed an adaptive hybrid algorithm called ASA-GS by combining the problem solving efforts of the SA algorithm and greedy search (GS) mechanism to solve the symmetric TSP problem [6]. They use a greedy search strategy in the optimization procedure of general SA algorithm to speed up the solution convergence rate. Their testing experiments demonstrated that the ASA-GS performs well on both small and large-scale TSP datasets and even outperforms some recently developed optimization algorithms. In fact, this composite algorithm found a better trade-off between solution quality and computation time for solving symmetric TSP problem. Utilize the benefit of the genetic algorithm (GA), SA, ant colony optimization (ACO) and particle swarm optimization (PSO) a hybrid algorithm named GSA-ACO-PSO is reported by Chen and Chien to tackle symmetric TSP problem [7]. In this optimization framework, a set of feasible solutions are generated by the ant colony system and these feasible solutions are adopted as the initial solution of the GA procedure. Then, the GA is executed with SA mutation techniques to achieve better solutions. On the other hand, the role of the particle swarm optimization process is to facilitate the exchange of pheromone information among the populations in the ant colony system after a predefined number of cycles. The experimental evaluations indicated that this hybrid algorithm exhibits better performance than some other related optimization algorithms.

Deng *et al.* developed a hybrid algorithm with the help of evolutionary concepts of GA, ACO and PSO algorithms to solve the symmetric TSP problem [3]. In the implementation process of this algorithm, a series of sub-optimal solutions are first generated through the combination of wholeness, randomicity, and rapidity of the PSO and GA techniques. After that, the resulting solutions are exploited based on the ACO algorithm by utilizing the benefit of the parallel, positive feedback and higher accuracy. Their algorithm evaluation illustrated that it performs better than some other evolutionary TSP solving algorithms. In [8], Zhan *et al.* proposed a new version of the SA algorithm named list-based SA algorithm (LBSA) in order to solve the symmetric TSP problem. In this approach, a list-based cooling schedule is adopted to control temperature reduction in the basic SA algorithm. Their experimental results indicated that the LBSA finds good approximate solutions and outperforms some other state-of-the-art algorithms. A hybrid optimization algorithm by combining the superiority of the symbiotic organism search (SOS) and SA algorithms named SOS-SA is reported by Ezugwu *et al.* [9]. In this optimization framework, the initial feasible solutions for the SOS algorithm are generated by applying the conventional SA algorithm. After that, these solutions are modified and improved through the three intelligent optimization phases of SOS algorithm. Comparative results demonstrated that the SOS-SA framework can yield TSP optimal solutions and show competitive behavior with other state-of-the-art optimization algorithms.

Hore *et al.* reported a hybrid variable neighborhood search (HVNS) algorithm

to solve both symmetric and asymmetric TSPs [4]. They accomplish the search procedure by two stages such as it first generates an initial feasible solution through a route construction based greedy approach, and then improves this solution iteratively by using various neighborhood structure with stochastic stopping criteria. The algorithm evaluation found that it performs better than the conventional optimization algorithms, and VNS-1 and VNS-2 algorithms as well. In [10], Tsai *et al.* proposed a hybrid algorithm called ACOMAC by introducing multiple ant clans (MAC) idea in the process of ACO algorithm. They also discussed ant colony system (ACS) for solving the TSP problem. In this work, multiple nearest neighbor (NN) and dual nearest neighbor (DNN) are combined separately with both ACOMAC and ACS to enhance their performance. Their experiments analysis indicated that the performance of both the basic ACOMAC and ACS are enhanced significantly when they combine with NN and DNN, meanwhile, ACOMAC + DNN performs better than the other discussed algorithms.

3. Methods of Study

In this section, we give a brief introduction of the optimization techniques under consideration in this study that make up the proposed hybrid algorithm for solving TSP problem. The RNN optimization algorithm and the SA optimization algorithm are briefly discussed in the following subsections consecutively.

3.1. Repetitive Nearest Neighbor Optimization Algorithm

The RNN is the route construction algorithm that is an extension of the well-known Nearest Neighbor (NN) algorithm [11]. The NN algorithm attempts to construct the route based on the connections of nearest neighbors. It starts with a city chosen at random as the starting city of the route and then includes the next city which is located closest to the last city. The performance of this algorithm is highly sensitive to the choice of starting city. To remedy this, the RNN algorithm was developed. It performs better than the NN algorithm for solving TSP problem. However, there is time complexity of order $O\left(n^3\right)$ while the algorithm running time of NN is reported as $O\left(n^2\right)$ [12]. In fact, the RNN algorithm constructs a set of routes step by step through some strategies. Its route construction procedure can be described as a sequence of the following steps.

Step-1: Let $C=\left\{c_1,c_2,c_3,\cdots,c_n\right\}$ be the list of n cities and $d\left(c_i,c_j\right)$ be the cost/distance between the cities c_i and c_j, where c_i represents the position of the i^{th} city. In the first step, the search engine generates n sub-routes and each sub-route consists of one single city. The set of constructed sub-routes can be expressed as follows:

$$R_1=\left\{c_i;i=1,2,\cdots,n\right\} \tag{6}$$

Step-2: Each sub-route obtained from the first step is extended the network in this step by adding a different nearest city from the remaining cities. Thus, the set of possible n sub-routes is constructed on the basis of Equation (7).

$$R_2 = \left\{ c_i c_j ; \forall c_i \in R_1 ; \min d\left(c_i, c_j\right), \forall c_j \in C; i \neq j; j = 1, 2, \cdots, n \right\} \quad (7)$$

In the Equation (7), $d\left(c_i, c_j\right)$ is the Euclidean distance between the cities c_i and c_j, which is calculated based on the formula presented in Equation (8). Let $\left(x_i, y_i\right)$ and $\left(x_j, y_j\right)$ be the Cartesian coordinate of the location of the cities c_i and c_j. Then, the formula for calculating the Euclidean distance between c_i and c_j is as follows:

$$\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \quad (8)$$

Step-3: In this step, each 2-city sub-route of R_2 is enlarged through a nearest city that is not yet in the route. Thus, the set of possible 3-city sub-routes is constructed as bellows:

$$R_3 = \left\{ c_i c_j c_k ; \forall c_i\, c_j \in R_2 ; \min d\left(c_j, c_k\right), \forall c_k \in C; i, j \neq k; k = 1, 2, \cdots, n \right\} \quad (9)$$

where $\left|R_3\right| = n$ is the total number of routes in the set R_3. In this way, the route construction mechanism is continued until all the cities are included in each route. After completing the n^{th} step, we get a set of routes R_n, where $\left|R_n\right| = n$ and each route of R_n contains n cities. To generate feasible TSP routes, we add the starting city to the $(n+1)^{\text{th}}$ position of each route in R_n. Finally, we get a set of feasible n routes and from there a best route is searched out. In this paper, the RNN method is considered to generate a set of n feasible TSP routes.

3.2. Simulated Annealing Optimization Algorithm

The SA algorithm for solving the TSP problem was first introduced by Kirkpatrick *et al.* in 1983 [13]. It is designed based on the idea of annealing process of metal atoms to achieve low energy states in a heat bath. Actually, metal atoms become unstable from their initial states at high temperature and they explore for other states. They find a lower energy state compared to their current state at the time of cooling. Thus, the procedure consists of two steps. First, the temperature of the heat bath is increased to a maximum value such that the metal atoms melt. Then, the temperature is reduced so that the particles cool until they are reached to a steady state. With regard to the search process, this algorithm consists of two important strategies-diversification and intensification. Setting the initial temperature high allow the exploration of the search space (diversification). On the other hand, the particles cool themselves until they converge into a steady state which means the search process converge to a local minimum (intensification). To maintain a proper balance between diversification and intensification strategies, a suitable temperature cooling schedule is required in this algorithm. We use an optimized cooling schedule in this paper which is defined in Equation (12).

The SA algorithm starts with randomly generating a set of initial solutions. The new candidate solutions/states are generated based on the initial solutions and the specified neighborhood structure. The neighborhood structure for generating new solution is briefly discussed in Subsection 3.2.1. The new solution

is accepted by the algorithm when its energy/fitness value is lower than the current solution. On the other hand, a non-improving new solution is accepted based on the transition probability *p*, defined as follows:

$$p = \mathrm{e}^{\left(-\frac{\Delta f}{T_k}\right)}, \tag{10}$$

$$\Delta f = \frac{f\left(x_i'\right) - f\left(x_i\right)}{f\left(x_i\right)}, \tag{11}$$

where $T_k \succ 0$ is the temperature in the k^{th} iteration, $f\left(x_i\right)$ and $f\left(x_i'\right)$ represent the fitness value of the current route and new route, respectively. If $\Delta f \le 0$, then the new solution x_i' is accepted. When $\Delta f > 0$, then the system changes the current solution x_i according to the probability *p*. It is noted from Equation (10) that the probability of accepting a bad solution is proportionally decreased with the temperature T as $\lim_{T\to 0} \mathrm{e}^{\left(-\frac{\Delta f}{T_k}\right)} = 0$. Therefore, a rigorous temperature cooling schedule plays a vital role for the performance of the SA algorithm. The following cooling schedule is utilized in this work to solve TSP problem:

$$T_{k+1} = \gamma T_k, \tag{12}$$

In the above equation, γ represents the cooling coefficient, which is also called the temperature reduction rate. The values of γ lies between 0 and 1. The process shows the best solution, when the temperature reaches to a predefined temperature or the maximum number of iterations is met. The procedure of SA algorithm is presented in the **Algorithm 1**.

Algorithm 1. Pseudocode of the SA algorithm.

Input: Distance matrix *D*, Initial temperature T_0, Cooling rate γ, Maximum iteration Maxit
Output: Best Solution
1: Generate *n* feasible routes randomly, $X = \{x_1, x_2, \cdots, x_n\}$, compute the Euclidean distance/fitness value of each route, $f(X) = \{f(x_1), f(x_2), \cdots, f(x_n)\}$, assign the value of Maxit and the initial value of T_0 and γ
2: For each iteration, it = 1 to maximum iteration
3: For each route x_i, $i = 1$ to *n*
4: Generate a new route x_i' by adopting neighborhood structure (described in Section 3.2.1)
5: Compute Euclidean distance/fitness value of the new route $f(x_i')$
6: Calculate Δf on the basis of Equation (11)
7: If $\Delta f \le 0$, then update the current route x_i by assigning $x_i \leftarrow x_i'$
8: If $\Delta f \succ 0$, then compute the acceptance probability *p* by using Equation (10). If $p \ge u$, then update the current route x_i by assigning $x_i \leftarrow x_i'$, where *u* is the random number between 0 and 1
9: End for
10: Decrease the temperature based on Equation (12)
11: Update the best solution ever found
12: End for

3.2.1. Neighborhood Structure of the SA Algorithm

In this paper, we use three neighborhood operators namely swap, reversion and inversion to generate new solution from the current existing solution. The probabilities of occurring swap, reversion and insertion are 0.5, 0.2, and 0.3, respectively, while the selection method is Roulette Wheel Selection. These sets of neighborhood operators are briefly discussed subsequently.

Swap: The swap operator randomly selects the position of two cities from a route x_i and exchanges the position of these two cities to create a new route x_i'. Let the two positions i and j with $i \neq j$ be randomly selected from the route x_i. Then, the swap operator generates a new route x_i' by interchanging the cities between the position i and j of x_i. For example, let $x_i = (5,7,1,2,4,3,6)$ be a feasible route consists of 7 cities. If second and sixth positions are selected for swap operation, then the corresponding cities 7 and 3 of x_i are exchanged and generates a new route $x_i' = (5,3,1,2,4,7,6)$. The swap operation procedure is depicted in Figure 1.

Reversion: The reversion operator firstly locates the position of two different cities in a route randomly and then reverses the local path between these two cities. Consider a route x_i and the two cut points i and j with $i \neq j$ on x_i. Then, the reversion operator inverse the cities between the position i and j of x_i and generates a new route x_i'. To explain the reversion process, consider a route x_i with 7 cities expressed by $x_i = (5,7,1,2,4,3,6)$. Let the position second and fifth are randomly selected in x_i, then reversion operator modifies the route x_i and creates a new route x_i' by reversing the local path (7, 1, 2, 4) between the cities 7 and 4. After reversion operation, the modified route can be written as $x_i' = (5,4,2,1,7,3,6)$. This process is displayed graphically in Figure 2.

Insertion: The insertion operator firstly picks up two positions i and j with $i \neq j$ on x_i randomly, and then the city of i^{th} position is inserted into the city of j^{th} back position. To illustrate more clearly of insertion operation procedure, let x_i be a 7-city route, which is represented by $x_i = (5,7,1,2,4,3,6)$. Suppose third and sixth positions are considered in x_i to perform the insertion operation. Then, the city 1 is placed behind the city 3 for producing a new route x_i'. After performing insertion operation on x_i, the modified route x_i' can be written as $x_i' = (5,7,2,4,3,1,6)$. The insertion mechanism is illustrated in Figure 3.

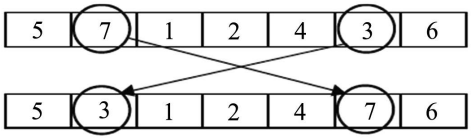

Figure 1. Illustration of the swap procedure [14].

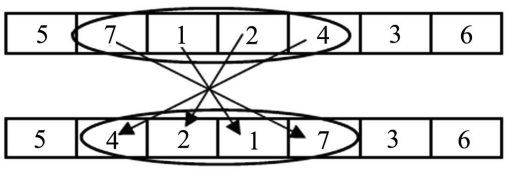

Figure 2. Illustration of the reversion procedure [14].

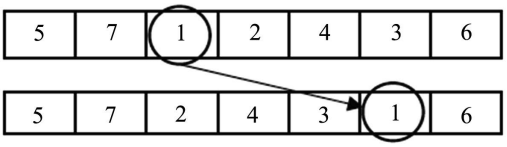

Figure 3. Illustration of the insertion procedure [14].

4. Proposed Hybrid Approach (RNN-SA)

In this section, we present a hybrid method called RNN-SA for the solution of symmetric TSP problem. The RNN algorithm is a simple and easily implementable constructive heuristic algorithm used to solve various optimization problems [11]. The main benefit of this algorithm is that it can produce a set of feasible solutions instead of one single solution. Logically, there is a more chance to get a better solution from a set of solutions instead of a single solution. In addition, The RNN algorithm performs better than the NN algorithm and yields the solution very promptly. However, its solution quality is not good enough compared to other algorithms. On the other hand, the SA algorithm is a local search based heuristic algorithm extensively utilized for finding the optimum solution of both discrete and continuous optimization problems [13]. One of the main advantages of this algorithm is that it accepts non-improving solutions with certain probability during the optimization process in the sense that a slightly worse solution than the current solution may provide a better solution in the near future. Indeed, the SA has the capability to prevent the search process from falling into the trapped of local optimum solution by permitting the uphill moves to search for a global optimum solution. The local optimum is a point in the search space where all the solution points in the neighborhood are worse than the current solution. However, the solution quality of the SA algorithm is still infected by the randomly initializing population. To overcome this problem, we adopt the RNN algorithm in the optimization process of the SA algorithm and enhance its performance. Therefore, a new hybrid algorithm (RNN-SA) is proposed by integrating the superiority of RNN and SA algorithms to effectively exploit and explore the problem search space.

Suppose that there are n cities in the TSP problem. The proposed algorithm accomplishes the search procedure by two stages. First, the RNN algorithm is used to generate a set of n feasible routes step by step and these routes are fed as the initial solution of the SA algorithm. Then, the system performs the SA algorithm to obtain better solutions by adopting a robust search process through a proper balance of diversification and intensification strategies. In fact, the feasible solutions obtained from the RNN algorithm are iteratively improved through the SA algorithm in an iterative improvement process. In each iteration, a set of neighborhood operators with their corresponding occurring probability are utilized to generate a new solution from the current solution. The selected neighborhood operator performs n moves to generate the new solutions in each iteration corresponding to each current solution. Thus, the proposed algorithm starts with generating a set of better routes by RNN and then adopts the SA algorithm

to improve these routes iteratively. It stops when the maximum number of iteration is achieved. The algorithm stopping condition is an important factor that can determine the final result of the experiment. If the algorithm is stopped too early, then the quality of the solution might not be even close to the best-known optimum solution. In contrast, prolonging the simulation incurs a considerable amount of unnecessary effort and time. So, it is crucial to consider the trade-off between the quality of solution and computational time. In this paper, we set up a fixed iteration number 1000 which is enough to get a satisfactory solution. The pseudocode and the architecture of the proposed hybrid optimization algorithm (RNN-SA) are offered in **Algorithm 2** and **Figure 4**, respectively.

5. Experiments and Analysis

In this section, we evaluate the performance of the proposed hybrid RNN-SA algorithm through various experiments. To accomplish the experimental tasks, a set of real-world symmetric TSP datasets are considered from TSPLIB [15]. Indeed, TSPLIB is a publicly available library that provides diverse testing datasets for the combinatorial optimization problems. It also reports the best-known optimum solution for each dataset. We consider 24 benchmark symmetric TSP datasets with dimension ranging from 51 to 318. The numerical value of the dataset name represents the dimension of that dataset. The experimental results of RNN-SA are first compared with both basic RNN and SA algorithms as well as the best-known optimum results reported by the dada library. Then, it is compared

Algorithm 2. Pseudocode of the proposed hybrid (RNN-SA) algorithm.

Input: Distance matrix D, Initial temperature T_0, Cooling rate γ, Maximum iteration Maxit
Output: Best Solution
1: Assign the value of Maxit and the initial value of T_0 and γ
2: For each city $c_i; i = 1, 2, \cdots, n$, generate a set of n sub-routes consist of one city c_i
3: Add a closest city in the right end of each sub-route that is not yet in the route
4: Repeat the process of step 3 until all the cities are added in each sub-route
5: Add starting city in the right side of the last position of each sub-route to generate n feasible routes, $X = \{x_1, x_2, \cdots, x_n\}$
6: Compute the Euclidean distance/fitness value of each route, $f(X) = \{f(x_1), f(x_2), \cdots, f(x_n)\}$
7: For each iteration, it = 1 to maximum iteration
8: For each route x_i, $i = 1$ to n
9: Generate a new route $x_i^{'}$ by adopting neighborhood structure (described in Section 3.2.1)
10: Compute Euclidean distance/fitness value of the new route $f(x_i')$
11: Calculate Δf on the basis of Equation (11)
12: If $\Delta f \le 0$, then update the current route x_i by assigning $x_i \leftarrow x_i'$
13: If $\Delta f \succ 0$, then compute the acceptance probability p by using Equation (10). If $p \ge u$, then update the current route x_i by assigning $x_i \leftarrow x_i'$, where u is the random number between 0 and 1
14: End for
15: Decrease the temperature based on Equation (12)
16: Update the best solution ever found
17: End for

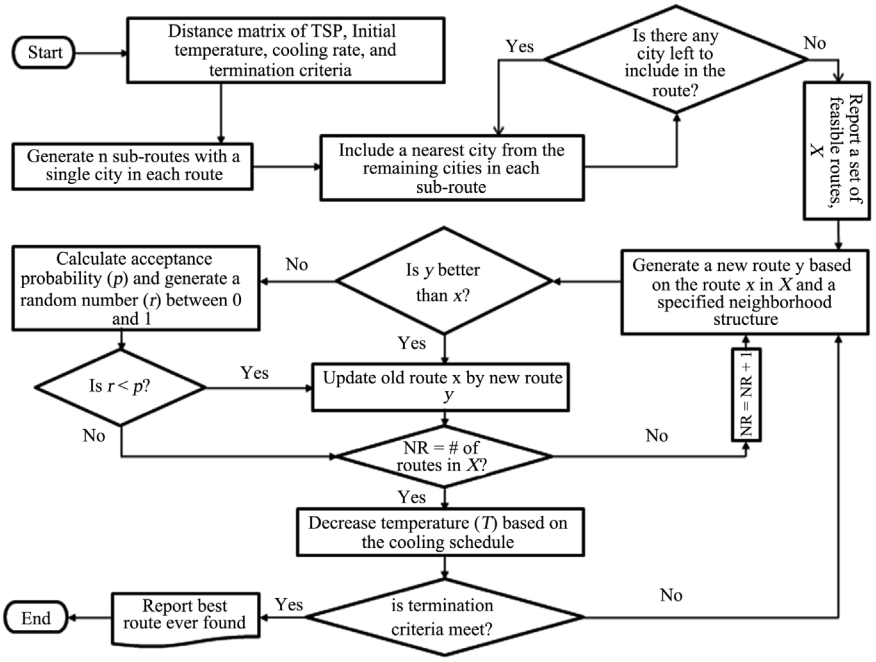

Figure 4. Architecture of the proposed hybrid optimization algorithm (RNN-SA).

with the results of some other hybrid algorithms existing in the literature. The results of various experiments might be different for same TSP dataset due to owing randomness in the optimization process of our RNN-SA algorithm. For this reason, the algorithm runs independently 5 consecutive times for each TSP dataset and from these 5 values the best solution, average solution, worst solution, and the standard deviation (SD) value are reported. Both basic algorithms (RNN and SA) and proposed (RNN-SA) algorithm are coded in MATLABR2017a programming language and executed on a standard laptop computer with Intel(R) Core(TM) i5-5337U CPU 1.80 GHz, 4 GB RAM and Windows 10 operating system. The maximum iteration for both SA and RNN-SA is set up as 1000 and in each run, the best solution from 1000 iterations are taken for a particular TSP dataset. The values of the parameters are adjusted through the experiment by trial and error method and their optimized values for both SA and RNN-SA algorithms are presented in Table 1.

5.1. Experimental Results of the Proposed RNN-SA Algorithm

The experimental results of the proposed RNN-SA algorithm for 24 benchmark symmetric TSP datasets are presented in Table 2. In Table 2, Scale represents the number of cities (dimension) of the dataset, BKS denotes the best-known optimum solution reported by the data library. Best, Average and Worst indicate the best optimum value, average optimum value, and worst optimum value of our algorithm among 5 optimum values. SD carries the standard deviation value computed based on the 5 optimum values. PDbest (%) and PDav (%) denote the percentage deviation of the computed best solution and average solution with respect to BKS. PDbest (%) and PDav (%) are computed according to Equation

Table 1. List of parameters and their optimized values for both SA and RNN-SA algorithms.

S/N	Parameter	Value
1	Size of population	n
2	Maximum Iteration	1000
3	Number of moves	n
4	Initial Temperature	0.025
5	Cooling Coefficient	0.99
6	Run	5
7	Swap Probability	0.2
8	Reversion Probability	0.5
9	Insertion Probability	0.3

Table 2. Experimental results of the proposed RNN-SA optimization algorithm for the 24 symmetric benchmark TSP datasets.

S/N	Datasets	Scale	BKS	Proposed Algorithm (RNN-SA)					
				Best	Average	Worst	SD	PDbest (%)	PDav (%)
1	eil51	51	426	428.87	429.44	430.24	0.52	0.673709	0.807512
2	berlin52	52	7542	7544.37	7544.37	7544.37	0.00	0.031424	0.031424
3	brazil58	58	25,395	25,395	25,440.40	25,622.00	9.39	0	0.178775
4	st70	70	675	677.11	679.33	682.66	3.04	0.312593	0.641481
5	eil76	76	538	544.37	549.16	555.99	4.40	1.184015	2.074349
6	rat99	99	1211	1219.24	1229.29	1232.68	5.68	0.680429	1.510322
7	kroA100	100	21,282	21,285.44	21,285.44	21,285.44	0.00	0.016164	0.016164
8	kroE100	100	22,068	22,119.90	22,164.92	22,267.17	58.65	0.235182	0.439188
9	eil101	101	629	644.92	652.31	657.71	5.39	2.531002	3.705882
10	bier127	127	118,282	119,331.15	119,849.69	120,061.28	294.00	0.88699	1.325383
11	ch130	130	6110	6171.87	6249.18	6331.14	66.31	1.012602	2.277905
12	pr136	136	96,772	96,922.41	100,335.16	101,924.34	1979.60	0.155427	3.682015
13	ch150	150	6528	6552.30	6553.94	6556.01	1.58	0.372243	0.397365
14	kroA150	150	26,524	26,821.83	27,008.24	27,330.04	194.18	1.12287	1.825667
15	kroB150	150	26,130	26,327.09	26,577.15	26,839.20	232.5	0.754267	1.711251
16	pr152	152	73,682	73,843.09	74,669.58	76,049.93	836.59	0.218629	1.340327
17	u159	159	42,080	42,162.75	42,547.66	42,715.47	90.57	0.196649	1.111359
18	rat195	195	2323	2348.70	2361.50	2375.73	11.49	1.106328	1.65734
19	d198	198	15,780	15,852.33	15,896.40	15,945.10	34.47	0.458365	0.737643
20	kroA200	200	29,368	29,541.83	29,626.42	29,702.88	73.01	0.591903	0.879937
21	kroB200	200	29,437	29,825.16	30,033.03	30,247.70	163.02	1.318613	2.024765
22	pr264	264	49,135	49,197.32	49,375.75	49,779.63	232.85	0.126834	0.489977
23	pr299	299	48,191	48,811.49	49,003.93	49,137.65	154.58	1.287564	1.686892
24	lin318	318	42,029	42,862.50	43,041.10	43,167.09	122.01	1.983154	2.707868
				Average				**0.71904**	**1.385866**

(13) and Equation (14). From the Table 2, it can be observed that the proposed RNN-SA algorithm yields the solutions very close to the BKS and the errors term are very small. Besides, the obtained SD values are small, which means that the proposed RNN-SA algorithm is stable to compute a satisfactory solution in each run. The errors lie within the interval of [0, 2.53] and [0.02, 3.68] with respect to the best and average solutions, respectively. In fact, the average errors over all the considered 24 datasets are 0.72 and 1.39 with respect to the best and average solutions. Therefore, it can be decided that the proposed RNN-SA algorithm is a reliable optimization algorithm that produces the solutions very near to the best-known optimum solutions.

$$\text{PDbest}(\%) = \frac{\text{Best} - \text{BKS}}{\text{BKS}} \times 100\% \tag{13}$$

$$\text{PDav}(\%) = \frac{\text{Average} - \text{BKS}}{\text{BKS}} \times 100\% \tag{14}$$

5.2. Performance Comparisons

In this subsection, we compare the performance of the proposed hybrid algorithm (RNN-SA) with the basic RNN and SA algorithms as well as some other hybrid algorithms. We code and execute the RNN-SA, RNN, and SA algorithms under the same experimental environment to show a fair comparison. For comparing with other hybrid algorithms, the results of RNN-SA are compared with the results presented in the reference articles. The side by side comparisons are displayed in Tables 3-5. The best results among the considered algorithms are highlighted in boldface. In addition, the average over all the considered datasets corresponding to each comparison is presented at the bottom of each table. Table 3 shows the comparison of RNN-SA for 24 symmetric TSP datasets with the basic RNN and SA algorithms. According to Table 3, the proposed RNN-SA algorithm shows better performance than both RNN and SA with respect to best, average as well as worst solutions in all the tested datasets except bier127 dataset. In bier127 dataset, our best result is inferior to SA but the average and worst results are superior to SA. Besides, the SD value of our algorithm is smaller than SA in nearly all the datasets. This indicator suggests that our algorithm (RNN-SA) is more stable than the SA algorithm. The average of best results, average results, worst results and SD values over all the 24 datasets of RNN-SA are 29,017.96, 29,295.97, 29,518.39, and 190.5763, respectively, which are better than the 29,303.13, 29,688.43, 29,994.45, and 306.0842 of SA as well as 33,137.39 of RNN.

The comparison of the proposed hybrid algorithm (RNN-SA) with the other hybrid algorithms (ACS + NN, ACOMAC + NN, ACS + DNN, and ACOMAC + DNN) is shown in Table 4. From Table 4, it is visible that the proposed RNN-SA algorithm is significantly dominated each of the other four hybrid algorithms in all the four datasets. In fact, the average over all the datasets of RNN-SA is 9527.753, which is better than the 9759.98 of ACS + NN, 9612.828 of ACOMAC + NN, 9691.583 of ACS + DNN, and 9586.613 of ACOMAC + DNN. Table 5 shows the performance comparison of RNN-SA with hybrid variable neighborhood

Table 3. Comparison of the proposed RNN-SA algorithm with the basic RNN and SA algorithms.

S/N	Datasets	Scale	BKS	Basic RNN	Basic SA				RNN-SA (Proposed)			
					Best	Average	Worst	SD	Best	Average	Worst	SD
1	eil51	51	426	505.77	428.98	430.99	432.71	1.70	**428.87**	**429.44**	**430.24**	**0.52**
2	berlin52	52	7542	8182.19	7544.37	7737.86	7914.29	135.08	7544.37	**7544.37**	**7544.37**	**0.00**
3	brazil58	58	25,395	27,384	25,601	25,617.80	25,622.00	101.52	**25,395**	**25,440.40**	**25,622.00**	**9.39**
4	st70	70	675	761.69	686.09	691.24	706.05	8.33	**677.11**	**679.33**	**682.66**	**3.04**
5	eil76	76	538	612.66	546.83	552.61	556.91	4.64	**544.37**	**549.16**	**555.99**	**4.40**
6	rat99	99	1211	1369.53	1219.86	1238.37	1268.71	23.43	**1219.24**	**1229.29**	**1232.68**	**5.68**
7	kroA100	100	21,282	24,698.49	21,285.44	21,460.80	21,690.57	190.41	21,285.44	**21,285.44**	**21,285.44**	**0.00**
8	kroE100	100	22,068	24,907.02	22,289.67	22,483.09	22,694.91	162.07	**22,119.90**	**22,164.92**	**22,267.17**	**58.65**
9	eil101	101	629	736.37	650.93	653.84	658.51	**3.17**	**644.92**	**652.31**	**657.71**	5.39
10	bier127	127	118,282	133,970.65	**118,846.22**	120,406.43	121,564.14	1125.70	119,331.15	**119,849.69**	**120,061.28**	**294.00**
11	ch130	130	6110	7198.74	6214.64	6253.37	6325.50	**43.09**	**6171.87**	**6249.18**	**6331.14**	66.31
12	pr136	136	96,772	114,560.90	98,853.84	100,607.01	101,424.68	**1028.30**	**96,922.41**	**100,335.16**	**101,924.34**	1979.60
13	ch150	150	6528	7078.44	6663.92	6691.84	6721.77	23.15	**6552.30**	**6553.94**	**6556.01**	**1.58**
14	kroA150	150	26,524	31,482.02	27,236.15	27,443.91	27,694.82	220.99	**26,821.83**	**27,008.24**	**27,330.04**	**194.18**
15	kroB150	150	26,130	31,320.34	26,641.66	27,054.47	27,669.88	398.52	**26,327.09**	**26,577.15**	**26,839.20**	**232.5**
16	pr152	152	73,682	79,566.56	74,251.63	74,683.95	74,972.16	**393.07**	**73,843.09**	**74,669.58**	**76,049.93**	836.59
17	u159	159	42,080	48,586.72	42,673.89	42,733.43	42,889.49	228.33	**42,162.75**	**42,547.66**	**42,715.47**	**90.57**
18	rat195	195	2323	2628.56	2381.90	2418.75	2459.95	30.96	**2348.70**	**2361.50**	**2375.73**	**11.49**
19	d198	198	15,780	17,809.73	15,913.5	16,042.91	16,130.76	86.10	**15,852.33**	**15,896.40**	**15,945.10**	**34.47**
20	kroA200	200	29,368	34,547.69	29,988.40	30,368.15	30,777.93	331.78	**29,541.83**	**29,626.42**	**29,702.88**	**73.01**
21	kroB200	200	29,437	35,394.01	30,329.79	30,567.35	30,841.37	231.92	**29,825.16**	**30,033.03**	**30,247.70**	**163.02**
22	pr264	264	49,135	54,491.48	49,434.52	51,479.29	52,840.02	1509.60	**49,197.32**	**49,375.75**	**49,779.63**	**232.85**
23	pr299	299	48,191	58,288.15	50,309.53	50,912.02	51,422.25	422.77	**48,811.49**	**49,003.93**	**49,137.65**	**154.58**
24	lin318	318	42,029	49,215.61	43,282.27	43,992.820	44,587.451	641.39	**42,862.50**	**43,041.10**	**43,167.09**	**122.01**
Average				33,137.39	29,303.13	29,688.43	29,994.45	306.0842	**29,017.96**	**29,295.97**	**29,518.39**	**190.5763**

Table 4. Comparison of the proposed RNN-SA algorithm with the ACS + NN (Tsai *et al.*, 2004), ACOMAC + NN (Tsai *et al.*, 2004), ACS + DNN (Tsai *et al.*, 2004), and ACOMAC + DNN (Tsai *et al.*, 2004) hybrid algorithms.

S/N	Datasets	Scale	BKS	ACS + NN [10]	ACOMAC + NN [10]	ACS + DNN [10]	ACOMAC + DNN [10]	RNN-SA (Proposed)
1	eil51	51	426	434.08	430.04	430.67	430.01	**428.87**
2	eil76	76	538	559.19	553.94	557.77	552.61	**544.37**
3	kroA100	100	21,282	21,487.95	21,433.33	21,440.09	21,408.23	**21,285.44**
4	d198	198	15,780	16,558.70	16,034	16,337.80	15,955.60	**15,852.33**
Average				9759.98	9612.828	9691.583	9586.613	**9527.753**

Table 5. Comparison of the proposed RNN-SA algorithm with the HVNS (Hore *et al.*, 2018) hybrid algorithms.

S/N	Datasets	Scale	BKS	HVNS (Hore *et al.*, 2018) [4]			RNN-SA (Proposed)		
				Best	Average	Worst	Best	Average	Worst
1	eil51	51	426	428.98	**428.98**	**428.98**	**428.87**	429.44	430.24
2	berlin52	52	7542	7544.37	7544.37	7544.37	7544.37	7544.37	7544.37
3	brazil58	58	25,395	25,425	25,592.72	25,664	**25,395**	**25,440.40**	**25,622.0**
4	st70	70	675	677.11	**677.11**	**677.11**	677.11	679.33	682.66
5	eil76	76	538	545.39	552.57	566.50	**544.37**	**549.16**	**555.99**
6	rat99	99	1211	1240.38	1241.26	1242.40	**1219.24**	**1229.29**	**1232.68**
7	kroA100	100	21,282	21,618.2	21,695.79	21,846.4	21,285.44	**21,285.44**	**21,285.44**
8	kroE100	100	22,068	22,174.6	22,193.8	22,222.36	**22,119.90**	**22,164.92**	**22,267.17**
9	eil101	101	629	**642.31**	**648.27**	657.91	644.92	652.31	**657.71**
10	bier127	127	118,282	**11,8974.6**	**119,006.39**	**119,054.4**	119,331.15	119,849.69	120,061.28
11	ch130	130	6110	**6140.66**	**6153.72**	**6165.14**	6171.87	6249.18	6331.14
12	pr136	136	96,772	97,979.11	**97,985.84**	**98,012.75**	**96,922.41**	100,335.16	101,924.34
13	ch150	150	6528	6639.52	6644.95	6666.66	**6552.30**	**6553.94**	**6556.01**
14	kroA150	150	26,524	26,943.31	**26,947.17**	26,962.6	**26,821.83**	27,008.24	27,330.04
15	kroB150	150	26,130	26,527.57	**26,537.04**	**26,576.12**	**26,327.09**	26,577.15	26,839.20
16	pr152	152	73,682	73,847.6	**73,855.11**	**73,885.15**	**73,843.09**	74,669.58	76,049.93
17	u159	159	42,080	42,436.23	**42,467.61**	**42,467.61**	**42,162.75**	42,547.66	42,715.47
18	rat195	195	2323	2450.14	2453.81	2464.32	**2348.70**	**2361.50**	**2375.73**
19	d198	198	15,780	16,075.84	16,079.28	16,085.53	**15,852.33**	**15,896.40**	**15,945.10**
20	kroA200	200	29,368	30,300.56	30,339.67	30,365.87	**29,541.83**	**29,626.42**	**29,702.88**
21	kroB200	200	29,437	30,447.30	30,453.22	30,472.42	**29,825.16**	**30,033.03**	**30,247.70**
22	pr264	264	49,135	51,155.38	51,197.14	51,364.2	**49,197.32**	**49,375.75**	**49,779.63**
23	pr299	299	48,191	50,271.69	50,373.12	50,778.86	**48,811.49**	**49,003.93**	**49,137.65**
24	lin318	318	42,029	43,924.08	43,964.93	44,128.35	**42,862.50**	**43,041.10**	**43,167.09**
Average				29,350.41	29,376.41	29,518.39	**29,017.96**	**29,295.97**	**29,429.17**

search (HVNS) algorithm for 24 symmetric TSP datasets. As shown in Table 5, the proposed RNN-SA algorithm exhibits competitive behavior with HVNS in terms of solution quality. It is observed that the HVNS finds better results in some small scale datasets. As the scale increases, our algorithm shows better global search capability than the HVNS algorithm. The average of best results, average results, and worst results over all the 24 datasets of RNN-SA are 29,017.96, 29,295.97, and 29,429.17, respectively which are better than the 29,350.41, 29,376.41, and 29,518.39 of HVNS.

6. Conclusion

We have established a hybrid heuristic algorithm called RNN-SA by integrating

the RNN algorithm and the SA algorithm for finding the optimum solution of the symmetric TSP problem. The proposed algorithm is composed of two main stages. In the first stage, we use the RNN algorithm to construct a set of feasible routes step by step. The concept of NN is adopted in this stage during the network enhancement process. In the second stage, we improve these routes in an iterative improvement process on the basis of the SA algorithm. Three neighborhood operators are utilized to generate new solution in the process of SA. Indeed, a set of feasible routes obtained from the RNN algorithm are fed in the SA algorithm to avoid random initialization and increase performance. The experiments are conducted with a set of benchmark symmetric TSP datasets taken from the TSPLIB. The experimental results demonstrate that the proposed hybrid (RNN-SA) algorithm yields the solutions very close to the best-known optimum solutions and performs better than both the basic algorithms (RNN and SA). In addition, the proposed hybrid optimization algorithm outperforms some other hybrid optimization algorithms existing in the literature in terms of solution quality.

Conflicts of Interest

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

References

[1] Applegate, D., Bixby, R., Cook, W. and Chvátal, V. (1998) On the Solution of Traveling Salesman Problems. *Documenta Mathematica*, Extra Volume of ICM, Chapter 3, 645-656.

[2] Dantzig, G., Fulkerson, R. and Johnson, S. (1954) Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, **2**, 393-410. https://doi.org/10.1287/opre.2.4.393

[3] Deng, W., Chen, R., He, B., Liu, Y., Yin, L. and Guo, J. (2012) A Novel Two-Stage Hybrid Swarm Intelligence Optimization Algorithm and Application. *Soft Computing*, **16**, 1707-1722. https://doi.org/10.1007/s00500-012-0855-z

[4] Hore, S., Chatterjee, A. and Dewanji, A. (2018) Improving Variable Neighborhood Search to Solve the Traveling Salesman Problem. *Applied Soft Computing*, **68**, 83-91. https://doi.org/10.1016/j.asoc.2018.03.048

[5] Garey, M.R. and Johnson, D.S. (2002) Computers and Intractability: Vol. 29. Wh Freeman, New York.

[6] Geng, X., Chen, Z., Yang, W., Shi, D. and Zhao, K. (2011) Solving the Traveling Salesman Problem Based on an Adaptive Simulated Annealing Algorithm with Greedy Search. *Applied Soft Computing*, **11**, 3680-3689. https://doi.org/10.1016/j.asoc.2011.01.039

[7] Chen, S.-M. and Chien, C.-Y. (2011) Solving the Traveling Salesman Problem Based on the Genetic Simulated Annealing ant Colony System with Particle Swarm Optimization Techniques. *Expert Systems with Applications*, **38**, 14439-14450. https://doi.org/10.1016/j.eswa.2011.04.163

[8] Zhan, S.H., Lin, J., Zhang, Z.J. and Zhong, Y.W. (2016) List-Based Simulated Annealing Algorithm for Traveling Salesman Problem. *Computational Intelligence and*

Neuroscience, **2016**, Article ID: 1712630. https://doi.org/10.1155/2016/1712630

[9] Ezugwu, A.E.-S., Adewumi, A.O. and Frîncu, M.E. (2017) Simulated Annealing Based Symbiotic Organisms Search Optimization Algorithm for Traveling Salesman Problem. *Expert Systems with Applications*, **77**, 189-210. https://doi.org/10.1016/j.eswa.2017.01.053

[10] Tsai, C.-F., Tsai, C.-W. and Tseng, C.-C. (2004) A New Hybrid Heuristic Approach for Solving Large Traveling Salesman Problem. *Information Sciences*, **166**, 67-81. https://doi.org/10.1016/j.ins.2003.11.008

[11] Rosenkrantz, D.J., Stearns, R.E. and Lewis II, P.M. (1977) An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing*, **6**, 563-581. https://doi.org/10.1137/0206041

[12] Johnson, D.S. and McGeoch, L.A. (1997) The Traveling Salesman Problem: A Case Study in Local Optimization. *Local Search in Combinatorial Optimization*, **1**, 215-310.

[13] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) Optimization by Simulated Annealing. *Science*, **220**, 671-680. https://doi.org/10.1126/science.220.4598.671

[14] Wang, Y., Wu, Y.W. and Xu, N. (2019) Discrete Symbiotic Organism Search with Excellence Coefficients and Self-Escape for Traveling Salesman Problem. *Computers & Industrial Engineering*, **131**, 269-281. https://doi.org/10.1016/j.cie.2019.04.008

[15] TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/