Improving variable neighbourhood search to solve the traveling salesman problem

Technical Report No. ASU/2017/15
Dated 18 September, 2017

Samrat Hore, Department of Statistics, Tripura University

Aditya Chatterjee, Department of Statistics, University of Calcutta

Anup Dewanji, Applied Statistics Unit, Indian Statistical Institute, Kolkata

Improving variable neighbourhood search to solve the traveling salesman problem

Samrat Hore, Department of Statistics, Tripura University

Aditya Chatterjee, Department of Statistics, University of Calcutta

Anup Dewanji, Applied Statistics Unit, Indian Statistical Institute, Kolkata

Abstract

The Traveling Salesman Problem (TSP) is one of the classical combinatorial optimization problems and has wide application in various fields of science and technology. In the present paper, we propose a new algorithm for solving the TSP, that uses the variable neighborhood search (VNS) algorithm coupled with a stochastic approach to find the optimal solution. Such neighborhood search with various other local search algorithms, named as $VNS-1$ and $VNS-2$, have been reported in the literature. The proposed algorithm is compared in detail with these algorithms, in the light of two benchmark TSP problems (one being symmetric while the other is asymmetric) suggested in the $TSPLIB$ data set in programming language R, along with two asymmetric problems obtained through simulation experiment. The present algorithm has been found to perform better than the conventional algorithms implemented in R for solving TSP's, and also, on an average, found to be more effective than the $VNS-1$ and the $VNS-2$ algorithms. The performance of the proposed algorithm has also been tested on 60 benchmark symmetric TSPs from the $TSPLIB$ data set.

Keywords: Neighborhood structure, Discrete optimization, Cost matrix, Stopping rule, Efficiency

1. Introduction

The TSP is a well-known, popular, and extensively studied problem in the field of combinatorial optimization. It has immense applications in the field of engineering science like designing hardware devices and radio electronic devices in communications, in the architecture of computational networks, designing of microchips, DNA sequencing, etc. [1]. The concept of the TSP is very simple but yet it remains one of the most challenging problems in operations research. More succinctly, it is an optimization problem to find the shortest connectivity among all the nodes in a closed network by touching all the nodes exactly once. In theoretical computer science, such kind of problem is known as the classical $NP-$complete problem, which means, for the worst-case, the running time for any algorithm for the TSP increases super-polynomially (or, perhaps exponentially) with the number of nodes [2]. In layman's language, the TSP is defined as follows: given n cities and the $n \times n$ cost matrix $(d(i, j))$ of visiting one city to another, how a salesman, starting from a home city, can arrange the tour consisting of the sequence of visits to all the cities, and return back to the home city, such that each city is visited exactly once and the total cost of the tour is minimum. The cost $d(i, j)$ of traveling from city i to j, for all $i \neq j$, may be distance, time, money, energy, etc.. It may be noted that the exact optimal tour may be found out by comparing the costs corresponding to all possible tours. However, the computer running time for this approach lies within a polynomial factor of $O(n!)$ and hence exact computation is almost impractical even for $n = 20$. Since 1950's, researchers are trying to develop various approximate algorithms to reach at a near-optimal solution. Graph theoretic method identifies the problem as finding the Hamiltonian cycle with the least weight for a given complete weighted graph [3].

As the exact optimal solution is seldom found for large n, various computer intensive metaheuristic algorithms have been proposed. The simplest of its kind is the Nearest Neighbor Algorithm (NNA) [4]. This algorithm starts with a sub-tour, or path, containing a subset of randomly chosen cities from the list of n cities and then adds the nearest city that is yet to visit. The algorithm stops when all cities are included in the tour. An extension to this algorithm is to repeat it with each city as the starting point and then return the best tour found, which is called the Repetitive Nearest Neighbor Algorithm ($RNNA$) [4]. Another approach is known as Insertion Algorithm (IA) [4], where it starts with a sub-tour consisting of two arbitrary cities (say, i and j) and then chooses in each step a city k not yet in the sub-tour. This city is inserted between the two cities i and j, such that the insertion cost (i.e., the net increase in the cost of the tour) given by $d(i, k) + d(k, j) - d(i, j)$ is minimum. This insertion process is repeated with the current sub-tour $i \rightarrow k \rightarrow j$ and a fourth city is inserted between either i and k, or k and j, with the objective of minimizing the net increase in cost of the next sub-tour. As before, the algorithm stops when all the cities are included in the tour. According to Hahsler and Hornik [5], four variants of IA are available, depending upon the way the city is to be inserted next is chosen, which are (1) Nearest Insertion (NI), (2) Farthest Insertion (FI), (3) Cheapest Insertion (CI) and (4) Arbitrary Insertion (AI), among which the first two are most popular. For the comparative analysis with the proposed algorithm, we have considered only the first two.

Some algorithms based on local improvements of the existing tour through simple tour modifications are also available in the literature. Such algorithms are specified in terms of a class of operations (exchanges or moves) that can be used to convert one tour into another by reducing the tour length until a tour is reached for which no such operation yields an improvement (i.e., a locally optimal tour). Among such simple local search algorithms, the most famous are $2-Opt$ and $3-Opt$ algorithms. The $2-Opt$ algorithm was developed by Croes [6], although it was earlier proposed by Flood [7]. The procedure deletes two edges of the tour, thus breaking it into two paths, and then reconnects those paths in the other possible way. A modified $2-Opt$ algorithm named as the *Quicker* $2-Opt$, was introduced in [8], by reducing search space for selected pair of links, instead of deleting and reconnecting all possible pairs of links in $2-Opt$ algorithm. In the $3-Opt$ algorithm [9], the exchange replaces up to three edges of the current tour. The idea of $2-Opt$ and $3-Opt$ algorithms may be extended to $k-Opt$ algorithms where the exchange replaces up to k edges of the current tour. This extension by Lin and Kernighan [10] does not use a fixed value for k for its $k-Opt$ moves but tries to find the best choice of k for each move.

To obtain a local optimal solution, the search operation is confined among the neighbors of an initial solution, called neighborhood, where the neighbors are iteratively obtained through a systematic change of the nodes of the initial tour. Through this change, and by means of local search, an algorithm has been developed that leads to one kind of metaheuristic algorithm, known as variable neighborhood search (VNS) algorithm [8]. The $VNS-1$ and the $VNS-2$ algorithms are recognized as such neighborhood search approach, where the local searches (i.e., improvements over the present solution) are carried out through $2-Opt$ and quicker $2-Opt$ algorithms, respectively.

The algorithm, as introduced in the present paper, is also motivated by the VNS concept where the improvement is done through an additional stochastic approach. Such an algorithm has been suggested by Hore et al. [11, 12] to find a near-optimal or optimal allocation design with different treatments to several experimental units with known covariate(s). The proposed algorithm is similar to the algorithm suggested in the context of the design issue with necessary modifications for enlarging the search space. Variants of such stochastic VNS algorithm may be found in the model selection approach for distributed fault detection in wireless sensor networks [13] and in deriving the optimum progressive censoring plan for life testing experiment in reliability theory [14].

The paper has been structured as follows. In Section 2, a brief review of the VNS algorithm has been done. In order to obtain a near-optimal tour, a moderately acceptable initial solution is needed. A procedure to obtain such initial solution is discussed in Section 3. Section 4 describes the proposed algorithm, based on the initial tour. A detailed analysis of two TSPs, one each for symmetric and asymmetric case, from the $TSPLIB$ data set of `R`, along with two more asymmetric TSPs generated through simulation experiments, are considered in Section 5. These examples establish the efficacy of the proposed algorithm in comparison to existing algorithms, as available and implemented in `R` and other VNS algorithms. The performance of the present algorithm is also judged in light of 60 benchmark symmetric TSPs from the $TSPLIB$ data set. The data sets and corresponding optimum solutions are available at [15]. The paper ends with some concluding remarks and scopes for further research.

2. Review of the VNS Algorithm

We start with a brief review of the variable neighborhood search (VNS) algorithm with special emphasis on $VNS-1$ and $VNS-2$, as proposed in [8]. Consider the situation where a finite number k_{max} of pre-selected ordered neighborhood structures is available and let the set of neighbors in the k^{th} neighborhood of the tour x be denoted by $\mathcal{N}_k(x)$, for $k = 1, 2, \cdots, k_{max}$. Note that the neighborhood search is permitted up to k_{max} number of times. The basic VNS may be summarized as follows and for details we refer to [8].

Initialization: Find an initial solution x; select the set of neighborhood structures $\mathcal{N}_k(x),\ k = 1, 2, \cdots, k_{max}$, that will be used in the search; choose a stopping condition;

Repeat the following until the stopping condition is met:

(1) *Set* $k \leftarrow 1$;

(2) *Until* $k = k_{max}$, *repeat the following steps :*

(a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$).

(b) *Local Search.* Apply some local search method with x' as the initial solution; denote the so obtained local optimum by x''.
(c) *Move or Not.* If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with $\mathcal{N}_1(x'')$ for $k \leftarrow 1$; otherwise set $k \leftarrow k + 1$.
Stopping Condition: Stop if maximum number of iterations is achieved.

The stopping condition may be based on the maximum amount of CPU time allowed, or the maximum number of iterations permitted, or the number of iterations between two significant improvements, and so on. In the simulation study and real life experiments, we have considered the maximum number of iterations as the stopping condition. The most popular version of local search algorithm for solving TSP is the $2-opt$ algorithm. The basic VNS rules using $2-Opt$ and $Quicker$ $2-Opt$ as local search method are known as $VNS-1$ and $VNS-2$, respectively. Thus, the only difference between $VNS-1$ and $VNS-2$ lies in the choice of the local search method, although the definition of neighborhood for both $VNS-1$ and $VNS-2$ are the same. Here $\mathcal{N}_k(x)$ is the k^{th} neighborhood of a tour x, for $k = 1, 2, \cdots, n-1$, which is obtained by randomly exchanging $(k+1)$ elements of x, i.e., having $(k+1)$ edges different from x. In the simulation study and real life examples, we have observed that $VNS-2$ consistently outperforms $VNS-1$ with varying extent.

There is scope for improvement in both $VNS-1$ and $VNS-2$ with respect to three important aspects, namely, (1) determination of the initial tour, (2) construction of neighborhood structure including the method of local search and (3) developing an efficient stopping rule. In the next two sections, we suggest some improvements in these three aspects and, in Section 5, illustrate them to find optimal or near-optimal tour by means of some real life examples and simulation studies.

3. The Initial Tour

We propose a greedy algorithm to obtain an initial solution, which is intuitively very simple and quite similar to NNA. In NNA, a tour on the basis of a subset of randomly chosen cities adds one city which is not included in the tour and is closest to the last city of the existing tour. Here we suggest that the tour should start from that particular sub-tour with two cities which has the least distance among all such possible sub-tours. Although such algorithm usually does not give the global optimum solution, we may consider it as the initial solution [16].

Let $I = \{1, 2, \cdots, n\}$ be the set of n distinct cities. The cost function between any two cities r and s is given by $d(r, s)$ with $d(r, r) = 0$, for $r, s \in I$. For symmetric TSP $d(r, s) = d(s, r)$, for all $r, s \in I$. Suppose $d(i, j) = min_{r,s}\{d(r, s);\ r, s \in I\}$ and assume this to be unique; if not, one can choose the minimum randomly. The tour should start from i to j, denoted by $\{i, j\}$. Let k be the nearest city with the tour $\{i, j\}$, in terms of the minimum cost $d(i, j, k)$ or $d(k, i, j)$ depending upon whichever is minimum, where $d(i, j, k) = min_l\{d(i, j) + d(j, l), l \in I, l \neq i, j\}$ or $d(k, i, j) = min_l\{d(l, i) + d(i, j), l \in I, l \neq i, j\}$, respectively. Then the city k is augmented with the already existing tour $\{i, j\}$ either at the beginning or at the end, and the augmented tour is denoted by either of $\{k, i, j\}$ or $\{i, j, k\}$, respectively. The procedure is continued till all the cities are included in the tour.

4. The Proposed Algorithm

Once the initial tour is judiciously obtained, an iterative algorithm is proposed to arrive at the optimal or near-optimal tour. Consider a tour given by a permutation $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ of $\{1, 2, \cdots, n\}$, the labels of n cities. Let us define the first order neighborhood of a tour α, denoted by $\mathcal{N}_1(\alpha)$, as $\{\alpha_{(11)}, \alpha_{(12)}, \cdots, \alpha_{(1n)}\}$, where $\alpha_{(1i)}$ is the permutation obtained from α by exchanging the elements at the i^{th} and $(i + 1)^{st}$ locations, i.e. α_i and α_{i+1}, respectively, for $i = 1, 2, ..., n$, with $\alpha_{n+1} = \alpha_1$, keeping other elements unchanged. Thus,

$$\begin{aligned}
\alpha_{(11)} &= (\alpha_2, \alpha_1, \alpha_3, ..., \alpha_{n-1}, \alpha_n),\\
\alpha_{(12)} &= (\alpha_1, \alpha_3, \alpha_2, ..., \alpha_{n-1}, \alpha_n),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\\
\alpha_{(1\ \overline{n-1})} &= (\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n, \alpha_{n-1}),\\
\alpha_{(1\ n)} &= (\alpha_n, \alpha_2, \alpha_3, ..., \alpha_{n-1}, \alpha_1).
\end{aligned}$$

The neighborhood construction for obtaining the neighbors of a selected initial tour may be generalized on the basis of this location exchanging concepts in subsequent orders of neighborhood. The kth order neighborhood of a tour α, denoted by $\mathcal{N}_k(\alpha)$, is constructed by exchanging α_i and $\alpha_{i+k\ (mod\ n)}$, for $i = 1, 2, ..., n$, while keeping other elements unchanged.

Note that $\mathcal{N}_i(\alpha) = \mathcal{N}_{n-i}(\alpha)$, for $i = 1, 2, ..., n$, so that maximum value of k is $k_{max} = [n/2]$. The cost of the tour corresponding to the permutation α is denoted by $V(\alpha)$, where $V(\alpha) = d(\alpha_1, \alpha_2) + d(\alpha_2, \alpha_3) + ... + d(\alpha_{n-1}, \alpha_n) + d(\alpha_n, \alpha_1)$. All the $n!$ possible tours are connected through this concept of neighborhood in the sense that any tour may be obtained from any another tour through a finite number of such exchange operations. The proposed algorithm for obtaining the optimum tour, starting from an initial tour, is described in the following steps along with explanation after each step.

Step 1: Start with the initial tour $\alpha^{(0)}$ with corresponding cost $V(\alpha^{(0)})$, say.

The initial tour is selected by the algorithm suggested in Section 3. The performance of the proposed algorithm depends upon a reasonable choice of the initial tour, failing which one may end up with one that is far from the optimal tour.

Step 2: Consider $k = 1$.

The exploration of neighborhoods starts with the first order neighbors of the initial tour.

Step 3 : Consider all the kth order neighbors in $\mathcal{N}_k(\alpha^{(0)})$, and compute $V(\alpha')$, for all $\alpha' \in \mathcal{N}_k(\alpha^{(0)})$.

Note that, for each k, a $n-$dimensional vector represents a neighbor and there are n such different neighbors. The cost for each of them is computed.

Step 4: If min $\{V(\alpha'), \alpha' \in \mathcal{N}_k(\alpha^{(0)})\} < V(\alpha^{(0)})$, choose the next improved tour to be $\alpha^{(1)} =$ arg min$\{V(\alpha'), \alpha' \in \mathcal{N}_k(\alpha^{(0)})\}$, and replace $\alpha^{(0)}$ by $\alpha^{(1)}$, to start search again from Step 2.

A better tour than the initial one is found from among the kth order neighbors and the algorithm starts afresh from Step 2, with the better tour replacing the initial one, to find a still better tour than the present one.

Step 5: Otherwise, set $k = k + 1$ and go to Step 3, till $k = [n/2]$.

If no better tour is found till the kth order, then extend the search domain to include the $(k+1)$st order neighbors of $\alpha^{(0)}$ and the search is continued till a better tour is obtained from among all the neighbors of the previous neighborhoods and including the present one.

Step 6: When $k > [n/2]$, check the stopping condition described in Step 7. If it is not satisfied, choose $\alpha^{(1)}$ randomly from $\alpha^{(0)} \cup \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})$ with probabilities given by $p_0 = \frac{V^{-1}(\alpha^{(0)})}{D(\alpha^{(0)})}$ at $\alpha^{(0)}$, $p_{\alpha'} = \frac{V^{-1}(\alpha')}{D(\alpha^{(0)})}$, for $\alpha' \in \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})$, where $D(\alpha^{(0)}) = V^{-1}(\alpha^{(0)}) + \sum_{\alpha' \in \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})} V^{-1}(\alpha')$. Set $\alpha^{(0)} = \alpha^{(1)}$

and go to Step 2.

If no better tour is found till Step 5 with $k = [n/2]$, a tour is selected randomly with preassigned probabilities from among all the tours already searched in all the $[n/2]$ neighborhoods, including the initial tour, and the algorithm is started afresh from Step 2. The preassigned probabilities are computed so as to put smaller chances of being selected as the next tour for those having larger cost. From the above described method, it is clear that the algorithm may get stuck at a local minimum, if there is no better alternative among the searched neighborhoods. A remedy in this regard may be suggested as to restart the algorithm with another initial tour chosen randomly from among all the searched neighborhoods. This may not entirely solve the problem of local optimum, but has been found to be effective for the search of the global optimum.

Step 7: The algorithm continues till a solution reached with some desired accuracy by satisfying some convergence criterion.

This accuracy may be achieved in several ways, for example, by limiting the number of returns, say 10, to a certain tour. In the present paper, we have generalized the approach mentioned in [11], by suggesting a probabilistic method, as follows. At the i^{th} return $(i \geq 1)$ to a tour $\alpha^{(0)}$, the preassigned probabilities p_0 and $p_{\alpha'}$, respectively, for choosing $\alpha^{(0)}$ and α' in $\mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})$, as described in Step 6 above, are modified as $p_0^{(i)} = \frac{V^{(i)}(\alpha^{(0)})}{D^{(i)}(\alpha^{(0)})}$, $p_{\alpha'}^{(i)} = \frac{V^{(i)}(\alpha')}{D^{(i)}(\alpha^{(0)})}$ with $D^{(i)}(\alpha^{(0)}) = V^{(i)}(\alpha^{(0)}) + \sum_{\alpha' \in \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})} V^{(i)}(\alpha')$, $V^{(i)}(\alpha^{(0)}) = V^{-1}(\alpha^{(0)}) + (D(\alpha^{(0)}) - V^{-1}(\alpha^{(0)})) \times \frac{i}{n^{[n/2]}}$, $V^{(i)}(\alpha^{'}) = max\,\{V^{-1}(\alpha^{'}) - (D(\alpha^{(0)}) - V^{-1}(\alpha^{(0)})) \times \frac{i}{n^{[n]}}, 0\}$, for $\alpha' \in \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})$. This increases the probability of choosing the same tour $\alpha^{(0)}$ at successive returns. We suggest stopping the algorithm when this probability exceeds a preassigned value close to unity, usually taken as 0.95 or 0.99.

The VNS algorithm for a minimization problem, as introduced in [8], is a descent method, while the proposed method is a descent-ascent method with the inclusion of Step 6. This is because, even if a new solution is worse than the present one, a better solution might be obtained from among all the neighbors of the worse tour in any subsequent step. The concept of such descent-ascent method is motivated from the well-known simulated annealing (SA) method [17], where the similarity lies in the selection of a new solution from among all its neighbors through some stochastic mechanism. The proposed algorithm, including the stochastic approach, may be summarized as follows.

Initialization: Obtain the initial tour $\alpha^{(0)}$ using the method of Section 3.

Repeat the following procedure until the stopping condition is achieved:

(1) *Set* $k \leftarrow 1$;

(2) *Until* $k = [n/2]$, *repeat the following steps :*

(a) *Construction of Neighborhood:*

Construct the kth order neighborhood $\mathcal{N}_k(\alpha^{(0)})$ of $\alpha^{(0)}$, and compute the cost $V(\alpha^{'})$ corresponding to the tour α', for $\alpha' \in \mathcal{N}_k(\alpha^{(0)})$

(b) *Exploration of Neighborhood:*

Find the best neighbor $\alpha^{(1)}$ of $\alpha^{(0)}$ by satisfying $V(\alpha^{(1)}) = arg\ min\{V(\alpha'),\ \alpha' \in \mathcal{N}_k(\alpha^{(0)})\}$

(c) *Move or Not :*

If the solution $\alpha^{(1)}$ is better than $\alpha^{(0)}$, move there by setting $\alpha^{(1)} \leftarrow \alpha^{(0)}$ and go to (1) to continue the search in $\mathcal{N}_1(\alpha^{(0)})$; otherwise set $k \leftarrow k + 1$.

(3) When $k > [n/2]$, check the stopping condition. If the condition is not satisfied choose a tour $\alpha^{(1)}$ randomly with preassigned probabilities among all the neighbors in $\alpha^{(0)} \cup \mathcal{N}_1(\alpha^{(0)}) \cup \cdots \cup \mathcal{N}_{[n/2]}(\alpha^{(0)})$; then set $\alpha^{(1)} \leftarrow \alpha^{(0)}$ and go to (1).

Stopping Condition:

Until the probability of returning to the same tour is above 0.95 or 0.99.

5. Illustrations

In this section, we investigate the performance of the proposed algorithm in comparison with the various other algorithms available in R. We consider two examples, namely USCA50 and ft53, for symmetric and asymmetric TSPs, respectively, from the $TSPLIB$ data set of R. We also consider two asymmetric TSPs with 10 and 50 cities, respectively, where the distances between the cities are simulated from an *exponential* distribution with mean = 25. The performance of the proposed algorithm is compared with NNA, $RNNA$, $2-Opt$, FI, NI, along with the $VNS-1$ and the $VNS-2$ algorithms by considering the two real life as well as the two simulated data sets, as described before. The efficiency of the proposed algorithm over any of the above seven algorithms is defined by the ratio of the values of the cost for the tour obtained by the particular competing algorithm and the same for the tour obtained by the proposed algorithm. Formally, the efficiency is defined as $\frac{V(\alpha^*)}{V(\alpha^P)}$, with α^* and α^P denoting the tours obtained by the competing and the proposed algorithm, respectively. The efficiency values may be multiplied by 100 to express them in percentage. As the proposed algorithm has an in-built stochastic component, the computation of the efficiency value is repeated 1000 times and the average, along with the minimum and maximum efficiency values, are reported, for each of the seven cases. Tables 1 and 2, respectively, present the findings of USCA50 and ft53 data sets, while Tables 3 and 4 report the findings from the simulation experiments on the basis of 10 and 50 cities, respectively.

For the USCA50 data set, which is an example of symmetric TSP, the mean efficiencies of the proposed algorithm with respect to the seven competing algorithms are greater than 100% indicating the superiority of the proposed algorithm over those. However, the gain in average efficiency of the proposed algorithm over the $2-Opt$, FI, $VNS-1$, $VNS-2$ algorithms are marginal compared to those over the rest. While comparing with these four algorithms, the minimum efficiency gain of the proposed algorithm over 1000 such repetitions falls marginally below 100% indicating that, in some situations, these algorithms may outperform the proposed one, albeit to a small extent. For the ft53 data set, which is an asymmetric TSP, the gain in average efficiency of the proposed algorithm over all the seven competing algorithms is found to be moderate to very high, with an increase by 9 to 58%. Moreover, the minimum efficiency values for all other competing algorithms, except the $VNS-2$ algorithm, are found to be larger than 100%. This indicates that, in the worst case also, the proposed algorithm possibly outperforms all except the $VNS-2$ algorithm, in which case also the efficiency loss is marginal. Thus the performance of the proposed algorithm over the seven competing algorithms may be claimed to be marginally to moderately superior for an asymmetric TSP.

For the two simulated examples of asymmetric TSP, the mean efficiency is never less than 100% and some times may be as high as 200%. Although, in some cases, the minimum efficiency in comparison to $2-opt$, $VNS-1$ and $VNS-2$ are less than 100%, but the loss in minimum efficiency is always marginal.

Table 1. Efficiency of the tours obtained by proposed algorithm with respect to the different algorithms over 1000 repetitions for USCA50 data.

Efficiency	Algorithms						
Measures	α^{NNA}	α^{RNNA}	α^{2-Opt}	α^{FI}	α^{NI}	α^{VNS-1}	α^{VNS-2}
Minimum	1.014	1.014	0.969	0.974	1.014	0.969	0.962
Mean	1.117	1.072	1.009	1.012	1.068	1.008	1.006
Maximum	1.523	1.237	1.124	1.138	1.194	1.053	1.053

Table 2. Efficiency of the tours obtained by proposed algorithm with respect to the different algorithms over 1000 repetitions for ft53 data.

Efficiency	Algorithms						
Measures	α^{NNA}	α^{RNNA}	α^{2-Opt}	α^{FI}	α^{NI}	α^{VNS-1}	α^{VNS-2}
Minimum	1.161	1.161	1.015	1.023	1.033	1.003	0.994
Mean	1.581	1.183	1.195	1.168	1.151	1.107	1.091
Maximum	2.099	1.319	1.244	1.217	1.293	1.162	1.153

Table 3. Efficiency of the tours obtained by proposed algorithm with respect to the different algorithms over 1000 repetitions for $n = 10$ randomly selected cities.

Efficiency	Algorithms						
Measures	α^{NNA}	α^{RNNA}	α^{2-Opt}	α^{FI}	α^{NI}	α^{VNS-1}	α^{VNS-2}
Minimum	1.131	1.131	0.972	1.021	1.093	0.951	0.939
Mean	1.734	1.524	1.325	1.381	1.512	1.010	1.007
Maximum	2.028	1.938	1.503	1.632	1.822	1.049	1.032

Table 4. Efficiency of the tours obtained by proposed algorithm with respect to the different algorithms over 1000 repetitions for $n = 50$ randomly selected cities.

Efficiency	Algorithms						
Measures	α^{NNA}	α^{RNNA}	α^{2-Opt}	α^{FI}	α^{NI}	α^{VNS-1}	α^{VNS-2}
Minimum	1.107	1.081	1.058	1.061	1.073	0.992	0.973
Mean	2.061	1.664	1.429	1.513	1.608	1.135	1.129
Maximum	2.352	1.837	1.634	1.719	1.814	1.417	1.328

Following Geng et al. [18], we also obtain the best, the average and the worst cost values out of 1000 replications of the proposed algorithm for some of the benchmark problems of the $TSPLIB$ data sets. An error in percentage for each instance, computed on the basis of the percentage of the deviation of the best value as obtained through the proposed algorithm from the optimum value as reported for the said data set, has also been computed. The corresponding CPU times (in second) have also been noted. The results are given in Table 5. We find that the loss of efficiency in the best solution, in comparison with the optimum one, as reported in the $TSPLIB$ data set, is mostly within 10% and seldom exceeds that, specifically for problems with larger number of cities. Moreover, in some instances, the proposed algorithm gives better solutions (for example, for $krob100$, $pr107$ and $pr144$ data sets) than those reported in the corresponding $TSPLIB$ data set, with negative error percentage entries in the table. The entire computation for the proposed algorithm is carried out in `R` software version 3.3.1 with a modest supporting system `AMD Phenom(tm) II N930 Quad-Core Processor 2.00 GHz` with 4.00 GB RAM in a standard laptop.

Table 5. Efficiency of the proposed algorithm for TSP benchmark instances from the $TSPLIB$ data set.

Sl No.	Instances	No. of cities	Optimum	Best	Average	Worst	Error (in %)	CPU time (in Sec)
1	gr17	17	2085	2085	2085	2085	0.00	16.51
2	gr21	21	2707	2707	2707	2707	0.00	18.38
3	gr24	24	1272	1272	1272	1272	0.00	23.14
4	fri26	26	937	937	937	937	0.00	39.57
5	bays29	29	2020	2020	2020	2020	0.00	48.29
6	dantzig42	42	699	699	699	699	0.00	151.38
7	swiss42	42	1273	1273	1273	1273	0.00	151.41
8	gr48	48	5046	5046	5046	5046	0.00	184.67
9	eil51	51	426	428.98	428.98	428.98	0.69	254.57
10	berlin52	52	7542	7544.36	7544.36	7544.36	0.03	261.32
11	brazil58	58	25395	25425	25592.72	25664	0.12	401.55
12	st70	70	675	677.11	677.11	677.11	0.31	532.61
13	eil76	76	538	545.39	552.57	566.50	1.37	614.28
14	pr76	76	108159	108159	108159	108159	0.00	614.36
15	rat99	99	1211	1240.38	1241.26	1242.40	2.42	828.57

Table 5 (Contd). Efficiency of the proposed algorithm for TSP benchmark instances from the $TSPLIB$ data set.

Sl No.	Instances	No. of cities	Optimum	Best	Average	Worst	Error (in %)	CPU time (in Sec)
16	rd100	100	7910	7910.4	7918.36	7930.39	0.00	937.68
17	kroa100	100	21282	21618.2	21695.79	21846.4	1.57	938.24
18	krob100	100	22141	22139.07	22140.20	22144.10	-0.009	938.26
19	kroc100	100	20749	20750.76	20809.29	20926.35	0.00	937.92
20	krod100	100	21294	21294.29	21490.62	21883.29	0.00	938.52
21	kroe100	100	22068	22174.6	22193.8	22222.36	0.48	937.83
22	eil101	101	629	642.31	648.27	657.91	2.11	941.39
23	lin105	105	14379	14383	14395.64	14412.75	0.00	962.47
24	pr107	107	44303	44301.68	44314.92	44324.84	-0.003	972.64
25	pr124	124	59030	59030.74	59051.82	59075.67	0.00	1029.57
26	bier127	127	118282	118974.6	119006.39	119054.4	0.58	1058.43
27	ch130	130	6110	6140.66	6153.72	6165.14	0.50	1082.19
28	pr136	136	96772	97979.11	97985.84	98012.75	1.24	1101.27
29	pr144	144	58537	58535.22	58563.97	58587.14	-0.003	1135.84
30	ch150	150	6528	6639.52	6644.95	6666.66	1.71	1201.63
31	kroa150	150	26524	26943.31	26947.17	26962.6	1.58	1201.49
32	krob150	150	26130	26527.57	26537.04	26576.12	1.52	1201.49
33	pr152	152	73682	73847.6	73855.11	73885.15	0.22	1234.27
34	u159	159	42080	42436.23	42467.61	42593.14	0.84	1281.73
35	rat195	195	2323	2450.14	2453.81	2464.32	5.47	1382.34
36	d198	198	15780	16075.84	16079.28	16085.53	1.87	1403.94
37	kroa200	200	29368	30300.56	30339.67	30365.87	3.17	1468.61
38	krob200	200	29437	30447.30	30453.22	30472.42	3.43	1468.61
39	pr226	226	80369	80469.31	80514.64	80695.97	0.01	1567.28
40	gil262	262	2378	2492.85	2501.86	2537.91	4.81	1596.73

Table 5 (Contd). Efficiency of the proposed algorithm for TSP benchmark instances from the $TSPLIB$ data set.

Sl No.	Instances	No. of cities	Optimum	Best	Average	Worst	Error (in %)	CPU time (in Sec)
41	pr264	264	49135	51155.38	51197.14	51364.2	4.03	1602.33
42	pr299	299	48191	50271.69	50373.12	50778.86	4.32	1687.52
43	lin318	318	42029	43924.08	43964.93	44128.35	4.51	1734.81
44	rd400	400	15281	16199.97	16250.21	16451.15	6.00	1953.49
45	fl417	417	11861	12180.78	12183.14	12192.56	2.69	2015.37
46	pr439	439	107217	111750.3	111771.2	111854.8	4.22	2106.84
47	pcb442	442	50778	50783.55	50800.24	50867	0.01	2183.27
48	U574	574	36905	39573.88	39629.11	39850.02	7.23	2351.39
49	rat575	575	6773	7349.81	7362.51	7413.27	8.52	2461.87
50	u724	724	41910	45725.39	45729.71	45746.97	9.10	3527.63
51	rat783	783	8806	9707.166	9707.364	9708.156	10.23	4028.14
52	pr1002	1002	259045	280368.2	280563.9	281346.7	8.23	6473.88
53	pcb1173	1173	56892	63354.82	63435.95	63760.47	11.36	9531.54
54	d1291	1291	50801	56088.31	56095.33	56123.41	10.40	11287.52
55	rl1323	1323	270199	295607.3	295611.2	295626.8	9.45	13421.56
56	fl1400	1400	20127	21040.65	21085.98	21085.98	4.53	15423.80
57	d1655	1655	62128	69992.49	70337.23	71716.2	12.65	18634.29
58	vm1748	1748	336556	366755.5	366757.8	366768.7	8.97	20314.86
59	u2319	2319	234256	252683.8	262595.6	252742.4	7.86	24534.72
60	pcb3038	3038	137694	154550.7	154565.4	154619.8	12.24	29412.80

6. Concluding Remarks

The present paper proposes a VNS algorithm with an inbuilt stochastic search approach for solving both symmetric as well as asymmetric TSPs, with equal ease of computation. While the performance of the algorithm seems to be marginally better than the presently available methods for symmetric problems, a significant improvement is observed for asymmetric cases. The average computation times for the proposed algorithm with the already mentioned computational strength for the 60 symmetric TSP benchmark problems are noted. All the standard algorithms are primarily developed to deal with symmetric TSPs and the asymmetric TSPs are reformulated as symmetric TSPs doubling the number of cities by introducing a dummy city corresponding to each city [19]. The main advantage of the proposed algorithm is its applicability to the asymmetric problems with equal flexibility, without converting the asymmetric problem into a symmetric one. The adaptive simulated annealing algorithm with greedy search ($ASA - GS$) for the TSP has been developed in [18] that has achieved better results with faster convergence for large-scale problems of the $TSPLIB$ data set. But, the method has been framed for only symmetric TSPs, while the proposed algorithm has been framed for a general TSP format. It is important to note that the present algorithm yields better solution for a few symmetric problems given in $krob100$, $pr107$, and $pr144$, in comparison to those reported in the $TSPLIB$ data set. A modified simulated annealing (SA) algorithm, named as list-based simulated annealing ($LBSA$) algorithm has been developed for solving symmetric TSPs in [20] by incorporating a selected list-based cooling schedule to obtain an optimal or near-optimal tour, with faster convergence rate. However, the $LBSA$ algorithm also fails to achieve better optimum values for the above three data sets [20].

Till date, it is generally believed that Concorde [21, 22] is possibly the best available algorithm for solving symmetric TSP's based on the Branch-and-Cut [23] and Branch-and-Bound [24] method. The optimal tour value of USCA50 and ft53 data from the $TSPLIB$ data set by using Concorde are reported in [5, 25], respectively. Although average performance of the proposed algorithm has been found to be marginally inferior to Concorde, but its performance has been found to be equally effective as Concorde in some replications. The detailed description and steps of the Concorde algorithm are not available in public domain and the software has to be installed separately, governed by a different license [22] and may not be compatible with all systems.

Applegate et al. [21] established new breakthrough results for the TSP. For instance, the research team solved an instance of the TSP of 13,509 cities corresponding to all US cities with populations of more than 500 people. The approach involved ideas from polyhedral combinatorics and combinatorial optimization, integer and linear programming, computer science data structures and algorithms, parallel computing, software engineering, numerical analysis, graph theory, and more. The solution of this instance demanded the use of three

Digital Alpha Server 4100s (with a total of 12 processors) and a cluster of 32 Pentium-II PCs. The complete calculation took approximately three months of computer time. The code had more than 1,000 pages and is based on state-of-the-art techniques from a wide variety of scientific fields [26]. This speaks of the volume of labor and effort needed to solve a large TSP. However, our method has the ability to tackle moderately large TSPs and to give relatively efficient solution by consuming very limited computer time in a standard laptop, without concerning about the problem being symmetric or asymmetric. At the same time, the present algorithm is intuitively very simple and may be judiciously implemented to solve various other problems of discrete optimization coming from different fields, as have been discussed [11, 12, 13, 14]. Some interesting extensions of the existing TSPs might be formulated and may be solved by suitably modifying the proposed algorithm in a constrained set up. One such extension might be assigning some infinitely large values in the cost matrix, thus blocking the corresponding paths in the tour. Another possible extension is to allow multiple visits to a selected group of cities, either as an option or as a mandate. The TSP with profits [27] may be an interesting extension, where it is not necessary to visit all the vertices and a profit is associated with each vertex. Extension of the algorithm to a muti-dimensional TSP, e.g. the spherical TSP, in which all the nodes (cities) and paths (solutions) are located on the surface of a sphere [28], might be an interesting problem. Works in these directions are already in progress and would be reported elsewhere.

Acknowledgment

The authors like to thank Michael Hahsler, Southern Methodist University, USA and Kurt Hornik, Wirtschaftsuniversitiat Wien, Austria for some helpful discussions and their `R` - guide notes 'TSP-Infrastructure for the Traveling Salesperson Problem and Package TSP'. The authors are also grateful to William Cook, Professor in Combinatorics and Optimization, University of Waterloo for his valuable suggestions and his reference at www.iwr.uni-heidelberg.de/groups/comopt/index.html.

References

[1] Z. C. S. S. Hlaing, M. A. Khine, Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm, International Journal of Information and Education Technology 1 (5) (2011) 404-409.

[2] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman : New York, 1979.

[3] R. L. Graham, *Handbook of Combinatorics*, MIT Press, 1995.

[4] D. J. Rosenkrantz, R. E. Stearns, M. L. I. Philip, An Analysis of Several Heuristics for the Traveling Salesman Problem, SIAM Journal on Computing 6(3) (1977) 563-581.

[5] M. Hahsler, K. Hornik, TSP - Infrastructure for the traveling salesperson problem, Journal of Statistical Software 23 (2) (2007) 1-21.

[6] G. A. Croes, Method for Solving Traveling-Salesman Problems, Operations Research 6 (6) (1958) 791-812.

[7] M. M. Flood, The Traveling Salesman Problem, Operations Research 4 (1956) 61-75.

[8] P. Hansen, N. Mladenović, Variable neighborhood Search : Principles and Applications, European Journal of Operational Research 130 (2001) 449-467.

[9] S. Lin, Computer solutions of the traveling-salesman problem, Bell System Technology Journal 44 (1965) 2245-2269.

[10] S. Lin, B. Kernighan, An Effective Heuristic Algorithm for the Traveling Salesman Problem, Operations Research 21(2) (1973) 498-516.

[11] S. Hore, A. Dewanji, A. Chatterjee, Design issues related to allocation of experimental units with known covariates into two treatment groups, Journal of Statistical Planning and Inference 155 (2014) 117-126.

[12] S. Hore, A. Dewanji, A. Chatterjee, On Optimal Allocation of Experimental Units with Known Covariates into Multiple Treatment Groups, Calcutta Statistical Association Bulletin 68 (1-2) (2016) 69-81.

[13] M. Nandi, A. Dewanji, B. Roy, S. Sarkar, Model Selection Approach for Distributed Fault Detection in Wireless Sensor Networks, International Journal of Distributed Sensor Networks (2014), http://dx.doi.org. 10.1155/2014/148234.

[14] R. Bhattacharya, B. Pradhan, A. Dewanji, On optimum life testing plans under Type-II progressive censoring scheme using variable neighborhood search algorithm, TEST 25(2) (2014) 309-330, http://dx.doi.org.10.1007/s11749-015-0449-z.

[15] TSPLIB : https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (accessed 15 September 2017)

[16] J. Bang-Jensen, G. Gutin, A. Yeo, When the Greedy Algorithm fails, Discrete Optimization 1(2) (2004), 121-127.

[17] S. Kirkpatrick, C. D. Gelatt, M. Vecchi, Optimization by Simulated Annealing, Science 220 (4598) (1983), 671-680.

[18] X. Geng, Z. Chen, W. Yang, D. Shi, K. Zhao, Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search, Applied Soft Computing 11 (4) (2011) 3680-3689.

[19] R. Jonker, T. Volgenant, Transforming Asymmetric into Symmetric Traveling Salesman Problems, Operations Research Letters 2 (1983) 161-163.

[20] S. Zhan, J. Lin, Z. Zhang, Y. Zhong, List-Based Simulated Annealing Algorithm for Traveling Salesman Problem, Computational Intelligence and Neuroscience http://dx.doi.org/10.1155/2016/1712630, 2016.

[21] D. Applegate, R.E. Bixby, V. Chvatal, W. Cook, TSP Cuts Which Do Not Conform to the Template Paradigm. In M Junger, D Naddef (Eds.), Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions, 2241, Lecture Notes in Computer Science, Springer-Verlag, London, 2000, pp 261-304.

[22] Applegate, D., Bixby, R. E., Chvatal, V. and Cook, W. (2006): Concorde TSP Solver. http: //www.tsp.gatech.edu/Concorde/.

[23] M. Padberg, G. Rinaldi, Facet Identification for the Symmetric Traveling Salesman Polytope, Mathematical Programming 47(2) (1990) 219 -257.

[24] A. Land, A. Doig, An Automatic Method for Solving Discrete Programming Problems, Econometrica 28 (1960) 497-520.

[25] Best known solutions for asymmetric TSPs : http://comopt.ifi.uni-heidelberg.de /software/TSPLIB95/ATSP.html (accessed 15 September, 2017)

[26] P. Moscato, C. Cotta, *A Gentle Introduction to Memetic Algorithms* Handbook of Metaheuristics, Kluwer Academic Publishers, NY, 2003.

[27] D. Feillet, P. Dejax, M. Gendreau, Traveling Salesman Problems with Profits, Transportation Science 39 (2) (2005) 188-205.

[28] X. Chen, Y. Zhoua, T. Zhonghua, L. Qifang, A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical travelling salesman problems, Applied Soft Computing 58 (September) (2017) 104-114.