

Research Article

A Data-Guided Lexisearch Algorithm for the Asymmetric Traveling Salesman Problem

Zakir Hussain Ahmed

Department of Computer Science, Al-Imam Muhammad Ibn Saud Islamic University, P.O. Box 5701, Riyadh 11432, Saudi Arabia

Correspondence should be addressed to Zakir Hussain Ahmed, zhahmed@yahoo.com

Received 23 October 2010; Revised 15 March 2011; Accepted 26 April 2011

Academic Editor: Tamas Kalmar-Nagy

Copyright © 2011 Zakir Hussain Ahmed. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A simple lexisearch algorithm that uses path representation method for the asymmetric traveling salesman problem (ATSP) is proposed, along with an illustrative example, to obtain exact optimal solution to the problem. Then a data-guided lexisearch algorithm is presented. First, the cost matrix of the problem is transposed depending on the variance of rows and columns, and then the simple lexisearch algorithm is applied. It is shown that this minor preprocessing of the data before the simple lexisearch algorithm is applied improves the computational time substantially. The efficiency of our algorithms to the problem against two existing algorithms has been examined for some TSPLIB and random instances of various sizes. The results show remarkably better performance of our algorithms, especially our data-guided algorithm.

1. Introduction

The traveling salesman problem (TSP) is one of the benchmark and old problems in computer science and operations research. It can be stated as follows:

A network with n nodes (cities), "node 1" being the starting node, and a cost (or distance, or time etc.,) matrix $C = [c_{ij}]$ of order n associated with an ordered pair of nodes (i, j) is given. The problem is to find a least cost Hamiltonian cycle. That is, the problem is to obtain a tour $(1 = \alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n = 1) \approx \{1 \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_{n-1} \rightarrow 1\}$ representing a cyclic permutation for which the total cost

$$C(1 = \alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n = 1) \approx \sum_{i=0}^{n-1} c(\alpha_i, \alpha_{i+1}) \quad (1.1)$$

is minimum.

The TSP finds application in a variety of situations such as automatic drilling of printed circuit boards and threading of scan cells in a testable VLSI circuit [1], X-ray

crystallography [2], and so forth. On the basis of structure of the cost matrix, the TSPs are classified into two groups—symmetric (STSP) and asymmetric (ATSP). The TSP is symmetric if $c_{ij} = c_{ji}$, for all i, j and asymmetric otherwise. The TSP is a NP-complete combinatorial optimization problem [3]; and roughly speaking it means, solving instances with a large number of nodes is very difficult, if not impossible. Since ATSP instances are more complex, in many cases, ATSP instances are transformed into STSP instances and subsequently solved using STSP algorithms [4]. However, in the present study we solve the ATSP instances without transforming into STSP instances.

Since large size instances cannot easily be solved optimally by an exact algorithm, a natural question may arise that what maximum size instances can be solved by an exact algorithm. Branch and cut [5], branch and bound [6], lexsearch [7, 8] are well-known exact algorithms. In our investigation, we apply lexsearch algorithm to obtain exact optimal solution to the problem. The lexsearch algorithm has been successfully applied to many combinatorial optimization problems. Pandit and Srinivas [9] showed that lexsearch algorithm is better than the branch and bound algorithm. In lexsearch algorithm, lower bound plays a vital role in reducing search space, hence, reduces computational time. Also, preprocessing of data, before the lexsearch algorithm is applied, can reduce computational effort substantially [9, 10].

In this paper, we first present a simple lexsearch algorithm using path representation for a tour for obtaining exact optimal solution to the ATSP. Then a data-guided lexsearch algorithm is proposed by incorporating a data processing method to improve further the efficiency of the algorithm. Finally, a comparative study is carried out of our algorithms against lexsearch algorithm of Pandit and Srinivas [9], and results using integer programming formulation of Sherali et al. [11] for some TSPLIB and random instances of various sizes.

This paper is organized as follows: Section 2 presents literature review on the problem. A simple lexsearch algorithm is developed in Section 3. Section 4 presents a data-guided lexsearch algorithm for the problem. Computational experience for the algorithms has been reported in Section 5. Finally, Section 6 presents comments and concluding remarks.

2. Literature Review

The methods that provide the exact optimal solution to a problem are called exact methods. The brute-force method of exhaustive search is impractical even for moderate sized TSP instances. There are few exact methods which find exact optimal solution to the problem much more efficiently than this method.

Dantzig et al. [12] solved instances of the TSP by formulating as integer programming approach and found an optimal solution to a 42-node problem using linear programming (LP). Sarin et al. [13] proposed a tighter formulation for the ATSP that replaces subtour elimination constraints of Dantzig et al. [12] and solved five benchmark instances using branch and bound approach, however, as reported [13], it failed to provide competitive performance for the ATSP instances due to the size and structure of the LP relaxations. A class of formulations for the ATSP has been proposed by Sherali et al. [11], which is proved to be tighter than the formulation based on subtour elimination constraints of Sarin et al. [13]. Also, Öncan et al. [14] made a survey of 24 different ATSP formulations and discussed the strength of their LP relaxations and reported that the formulation by Sherali et al. [11] gave the tightest lower bounds.

Balas and Toth [15] solved the ATSP instances using a branch and bound approach with a bounding procedure based on the assignment relaxation of the problem. Currently,

many good approximation algorithms based on branch and bound approach have been developed for solving ATSP instances [16–18].

Pandit [19] developed a lexiseach algorithm for obtaining exact optimal solution to the ATSP by using adjacency representation for a tour. As reported, the algorithm shows large variations in the context of computational times for different instances of same size. Murthy [20] proposed another scheme for the “search” sequence of Pandit’s [19] algorithm, which was expected to increase the computational efficiency of the algorithm to considerable extent. But as reported by Srinivas [21], the proposed algorithm is likely to be efficient in case of highly skewed cost distributions and does not seem to be any better than the conventional branch and bound algorithm.

Pandit and Srinivas [9] again modified lexiseach algorithm of Pandit [19], which is found to be better than previous lexiseach algorithms and branch and bound algorithm. But, as reported by Ahmed [22], the algorithm shows large variations in computational times. It is interesting to see that randomly generated instances of same size seem to fall into two distinct groups in the context of computational time. One group requires significantly less time than the average while another takes significantly more time than the average, with a big “gap” between the two groups.

There are mainly two ways of representing salesman’s path in the context of lexiseach approach, namely, path representation and adjacency representation. In adjacency representation, permutation is generated in a systematic lexical order, but all permutations do not lead to feasible solution. Hence, a permutation is to be tested for acceptability. In path representation, explicit testing for cycle formation is avoided, and hence there is a possibility to take less computational time than the other method. In fact, Ahmed and Pandit [23] used path representation for solving the TSP with precedence constraints and found very good results. In this paper also we are using path representation method for a tour.

3. A simple Lexiseach Algorithm for the ATSP

In lexiseach approach, the set of all possible solutions to a problem is arranged in hierarchy-like words in a dictionary, such that each incomplete word represents the block of words with this incomplete word as the leader of the block. Bounds are computed for values of the objective function over these blocks of words. These are compared with “best solution value” found so far. If no word in the block can be better than the “best solution value” found so far, jump over the block to the next one. However, if the bound indicates a possibility of better solutions in the block, enter into the subblock by concatenating the present leader with appropriate letter and set a bound for the new (sub) block so obtained [7, 8, 10].

3.1. Bias Removal

The basic difference between Assignment Problem (AP) solution space and TSP solution space is that former can be viewed as the set of all permutations of n elements while the latter is restricted to “indecomposable” permutations only. So, solution space of TSP is a subset of solution space of AP and hence, steps like bias removal are useful in case of TSP [24].

For calculating bias, we first calculate row minima of the cost matrix C as

$$u_i = \min_{1 \leq j \leq n} \{c_{ij}\}, \quad \text{for } i = 1, 2, 3, \dots, n. \quad (3.1)$$

Table 1: The cost matrix with row minima and column minima.

Node	1	2	3	4	5	6	7	Row minima
1	999	75	99	9	35	63	8	8
2	51	999	86	46	88	29	20	20
3	100	5	999	16	28	35	28	5
4	20	45	11	999	59	53	49	11
5	86	63	33	65	999	76	72	33
6	36	53	89	31	21	999	52	21
7	58	31	43	67	52	60	999	31
Column minima	9	0	0	1	0	9	0	

Then we subtract each row minima u_i from its corresponding row elements in the matrix C to obtain a new matrix as

$$\forall j = 1, 2, 3, \dots, n, \quad \bar{c}_{ij} = c_{ij} - u_i, \quad \text{for each } i = 1, 2, 3, \dots, n. \quad (3.2)$$

Then we calculate column minima of the resultant cost matrix \bar{C} as

$$v_j = \min_{1 \leq i \leq n} \{\bar{c}_{ij}\}, \quad \text{for } j = 1, 2, 3, \dots, n. \quad (3.3)$$

Next, we subtract each column minima v_j from its corresponding column elements in the resultant matrix to obtain a modified matrix $\bar{\bar{C}}$ as

$$\forall i = 1, 2, 3, \dots, n, \quad \bar{\bar{c}}_{ij} = \bar{c}_{ij} - v_j, \quad \text{for each } j = 1, 2, 3, \dots, n, \quad (3.4)$$

$$\text{Bias of the matrix} = \sum_{i=1}^n u_i + \sum_{j=1}^n v_j. \quad (3.5)$$

The bias calculation is shown in Table 1. Bias of the given matrix = row minima + column minima = 129 + 19 = 148. The modified (resultant) cost matrix $\bar{\bar{C}}$ will become a nonnegative matrix with at least one zero in each row and in each column (see Table 2). In an AP, if we add or subtract a constant to every element of a row (or column) in the cost matrix, then an assignment which minimizes the total cost on one matrix also minimizes the total cost on the other matrix. Since, TSP is a particular case of AP, so, it is enough to solve the problem with respect to the modified cost matrix.

3.2. Alphabet Table

Alphabet matrix, $A = [a(i, j)]$, is a square matrix of order n formed by positions of elements of the modified cost matrix of order n , $C = [c_{ij}]$. The i th row of matrix A consists of the positions of the elements in the i th row of the matrix C when they are arranged in the nondecreasing order of their values. If $a(i, p)$ stands for the p th element in the i th row of A , then $a(i, 1)$

Table 2: The modified cost matrix (*diagonal elements need not have to be computed, which could as well be left as 999).

Node	1	2	3	4	5	6	7
1	982*	67	91	0	27	46	0
2	22	979*	66	25	68	0	0
3	86	0	994*	10	23	21	23
4	0	34	0	987*	48	33	38
5	44	30	0	31	966*	34	39
6	6	32	68	9	0	969*	31
7	18	0	12	35	21	29	968*

Table 3: The alphabet table (N is a node and V is its value).

Node	$N(V)$	$N(V)$	$N(V)$	$N(V)$	$N(V)$	$N(V)$	$N(V)$
1	4(0)	7(0)	5(27)	6(46)	2(67)	3(91)	1(982)
2	6(0)	7(0)	1(22)	4(25)	3(66)	5(68)	2(979)
3	2(0)	4(10)	6(21)	5(23)	7(23)	1(86)	3(994)
4	1(0)	3(0)	6(33)	2(34)	7(38)	5(48)	4(987)
5	3(0)	2(30)	4(31)	6(34)	7(39)	1(44)	5(966)
6	5(0)	1(6)	4(9)	7(31)	2(32)	3(68)	6(969)
7	2(0)	3(12)	1(18)	5(21)	6(29)	4(35)	7(968)

corresponds to the position of smallest element in i th row of the matrix C [7, 10]. Alphabet table " $[a(i, j) - c_{i,a(i,j)}]$ " is the combination of elements of matrix A and their values as shown in Table 3.

3.3. Lower Bound

The objective of lower bound is to skip as many subproblems in the search procedure as possible. A subproblem is skipped if its lower bound exceeds the "best solution" found so far (i.e., upper bound) in the process. The higher the lower bound the larger the set of subproblems that are skipped. Most of methods in the literature consider AP solution as the overall lower bound for the ATSP instance and develop algorithms based on AP relaxation and subtour elimination scheme. In this study we are not setting overall lower bound for an instance, rather setting lower bound for each leader on the value of the objective function for the instance. The following method is used for setting lower bound for each leader.

Suppose the partial tour is $(1 = \alpha_0, \alpha_1, \alpha_2)$ and "node α_3 " is selected for concatenation. Before concatenation, we check bound for the leader $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$. For that, we start our computation from 2nd row of the "alphabet table" and traverse up to the n th row, and sum up the values of the first "legitimate" node (the node which is not present in the tour), including "node 1", in each row, excluding α_1 -th and α_2 -th rows. This sum is the lower bound for the leader $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$.

3.4. Simple Lexisearch Algorithm (LSA)

Let $C = [c_{ij}]$ be the given $n \times n$ cost matrix. Then for all nodes in the network, generate a zero-one vector $V=[v_i]$ of order n as follows:

$$v_i = \begin{cases} 1 & \text{if "node } i" \text{ is in the tour (partial word),} \\ 0, & \text{otherwise.} \end{cases} \quad (3.6)$$

Though it is not a part of the algorithm, but for checking the feasibility of the tour, we follow this convention. A preliminary version of the algorithm is presented in Ahmed [22]. The algorithm is as follows.

Step 1. Remove "bias" of the cost matrix and construct the "alphabet table" based on the modified cost matrix. Set "best solution value" as large as possible. Since "node 1" is the starting node, we start our computation from 1st row of the "alphabet table". Initialize "partial tour value" = 0, $r = 1$ and go to Step 2.

Step 2. Go to r th element of the row (say, node p). If (partial tour value + present node value) is greater than or equal to the "best solution value", go to Step 10, *else*, go to Step 3.

Step 3. If "node p " forms a subtour, drop it, increment r by 1 and go to Step 8; *else*, go to Step 4.

Step 4. If all nodes of the network are visited, add an edge connecting the "node p " to "node 1", compute the complete tour value and go to Step 5; *else* go to Step 6.

Step 5. If the complete tour value is greater than or equal to the "best solution value", go to Step 10; *else*, replace the tour value as the "best solution value" and go to Step 10.

Step 6. Calculate the lower bound of the present leader on the objective function value and go to Step 7.

Step 7. If (lower bound + partial tour value + present node value) is greater than or equal to the "best solution value", drop the "node p ", increment r by 1, and go to Step 8; *else*, accept the "node p ", compute the partial tour value, and go to Step 9.

Step 8. If r is less than n (total number of nodes), go to Step 2; *else*, go to Step 10.

Step 9. Go to subblock, that is, go to p th row, put $r = 1$ and go to Step 2.

Step 10. Jump this block, that is, drop the present node, go back to the previous node in the tour (say, node q), that is, go to the q th row of the "alphabet table" and increment r by 1, where r was the index of the last "checked" node in that row. If node $q = 1$ and $r = n$, go to Step 11, *else*, go to Step 8.

Step 11. The "best solution value" is the optimal solution value with respect to the modified cost matrix. Add the "bias" to the optimal solution value to obtain the optimal solution value with respect to the original cost matrix, and go to Step 12.

Step 12. Current word gives the optimal tour sequence with respect to the original cost matrix, and then *stop*.

3.5. Illustration of the Algorithm

Working of the above algorithm is explained through a seven-node example with cost matrix given in Table 1. The logic flow of the algorithm at various stages is indicated in Table 4, which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps. The symbols used therein are listed below:

GS: go to sub-block,

JB: jump the current block,

JO: jump out to the next, higher order block,

BS: best solution value.

As illustration of the example, we initialize BS = 999 and “partial tour value (Sol)” = 0. We start from 1st row of the “alphabet table”. Here, $a(1, 1) = 4$ with “present node value (Val)” = $c_{14} = 0$ and $(\text{Sol} + \text{Val}) < \text{BS}$. Now we go for bound calculation for the present leader (1, 4). The bound will guide us whether the node 4 will be accepted or not

$$\begin{aligned} \text{Bound} &= c_{2,a(2,1)} + c_{3,a(3,1)} + c_{4,a(4,1)} + c_{5,a(5,1)} + c_{6,a(6,1)} + c_{7,a(7,1)} \\ &= c_{2,6} + c_{3,2} + c_{4,1} + c_{5,3} + c_{6,5} + c_{7,2} \\ &= 0 + 0 + 0 + 0 + 0 + 0 = 0. \end{aligned} \tag{3.7}$$

Since $(\text{Bound} + \text{Sol} + \text{Val}) = 0 + 0 + 0 = 0 < \text{BS}$, we accept the node 4 that leads to the partial tour $\{1 \rightarrow 4\}$ with $\text{Sol} = \text{Sol} + \text{Val} = 0 + 0 = 0$.

Next, we go to 4th row of the “alphabet table”. Since $a(4, 1) = 1$ forms to a sub-tour, we consider the next element of the row, $a(4, 2) = 3$ with $\text{Val} = c_{43} = 0$, and $(\text{Sol} + \text{Val}) < \text{BS}$. Now, we go for bound calculation for the present leader (1, 4, 3)

$$\begin{aligned} \text{Bound} &= c_{2,a(2,1)} + c_{3,a(3,1)} + c_{5,a(5,2)} + c_{6,a(6,1)} + c_{7,a(7,1)} \\ &= c_{2,6} + c_{3,2} + c_{5,2} + c_{6,5} + c_{7,2} \\ &= 0 + 0 + 30 + 0 + 0 = 30. \end{aligned} \tag{3.8}$$

Since, $(\text{Bound} + \text{Sol} + \text{Val}) = 30 + 0 + 0 = 30 < \text{BS}$, we accept the node 3 that leads to the partial tour $\{1 \rightarrow 4 \rightarrow 3\}$ with $\text{Sol} = \text{Sol} + \text{Val} = 0 + 0 = 0$. Proceeding in this way we obtain the 1st complete tour as $\{1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 1\}$ with $\text{Sol} = 57 < \text{BS}$, so we replace BS = 57. Now, we jump out to the next higher-order block, that is, $\{1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6\}$ with $\text{Sol} = 0$ and try to compute another complete tour with lesser tour value. Proceeding in this way, we obtain the optimal tour as $\{1 \rightarrow 7 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 1\}$ with value 10. Hence, the optimal tour value with respect to the given original cost matrix = bias + tour value = $148 + 10 = 158$.

Our preliminary study on randomly generated instances shows that the above algorithm also produces two groups of instances of same size in terms of computational

Table 4: The search table.

$1 \rightarrow \alpha_1$	$\alpha_1 \rightarrow \alpha_2$	$\alpha_2 \rightarrow \alpha_3$	$\alpha_3 \rightarrow \alpha_4$	$\alpha_4 \rightarrow \alpha_5$	$\alpha_5 \rightarrow \alpha_6$	$\alpha_6 \rightarrow 1$
$1 \rightarrow 4_{(0)}$ (0) + 0, GS	$4 \rightarrow 3_{(0)}$ (0) + 30, GS	$3 \rightarrow 2_{(0)}$ (0) + 52, GS	$2 \rightarrow 6_{(0)}$ (0) + 57, GS	$6 \rightarrow 5_{(0)}$ (0) + 57, GS	$5 \rightarrow 7_{(39)}$ (0) + 18, GS	$7 \rightarrow 1_{(18)}$ BS = 57, JO
				$6 \rightarrow 7_{(31)}$ (0) + 62, JO		
			$2 \rightarrow 7_{(0)}$ (0) + 52, GS	$7 \rightarrow 5_{(21)}$ (0) + 40, JB		
				$7 \rightarrow 6_{(29)}$ (0) + 44, JO		
			$2 \rightarrow 5_{(68)}$, JO			
		$3 \rightarrow 6_{(21)}$ (0) + 30, GS	$6 \rightarrow 5_{(0)}$ (21) + 39, JB			
			$6 \rightarrow 7_{(31)}$ (21) + 61, JO			
		$3 \rightarrow 5_{(23)}$ (0) + 40, JB				
		$3 \rightarrow 7_{(23)}$ (0) + 34, JO				
	$4 \rightarrow 6_{(33)}$ (0) + 0, GS	$6 \rightarrow 5_{(0)}$ (33) + 0, GS	$5 \rightarrow 3_{(0)}$ (33) + 0, GS	$3 \rightarrow 2_{(0)}$ (33) + 18, GS	$2 \rightarrow 7_{(0)}$ (33) + 18, GS	$7 \rightarrow 1_{(18)}$ BS = 51, JO
				$3 \rightarrow 7_{(23)}$, JO		
			$5 \rightarrow 2_{(30)}$, JO			
		$6 \rightarrow 7_{(31)}$, JO				
	$4 \rightarrow 2_{(34)}$ (0) + 33, JB					
	$4 \rightarrow 7_{(38)}$ (0) + 0, GS	$7 \rightarrow 2_{(0)}$ (38) + 21, JB				
		$7 \rightarrow 3_{(12)}$ (38) + 30, JB				
		$7 \rightarrow 5_{(21)}$, JO				
	$4 \rightarrow 5_{(48)}$ (0) + 6, JO					
$1 \rightarrow 7_{(0)}$ (0) + 0, GS	$7 \rightarrow 2_{(0)}$ (0) + 10, GS	$2 \rightarrow 6_{(0)}$ (0) + 10, GS	$6 \rightarrow 5_{(0)}$ (0) + 10, GS	$5 \rightarrow 3_{(0)}$ (0) + 10, GS	$3 \rightarrow 4_{(10)}$ (0) + 0, GS	$4 \rightarrow 1_{(0)}$ BS = 10, JO
				$5 \rightarrow 4_{(31)}$, JO		
			$6 \rightarrow 4_{(9)}$ (0) + 23, JB			
			$6 \rightarrow 3_{(38)}$, JO			
		$2 \rightarrow 4_{(25)}$, JO				
	$7 \rightarrow 3_{(12)}$, JO					
$1 \rightarrow 5_{(27)}$, STOP						

times with a gap between the groups. In the above algorithm, the nature of the data does not play any role. However, introducing a preprocessing of the cost matrix may reduce the computational time as well as gap between two groups, and hence, we introduce preprocessing technique in next section.

4. A Data-Guided Lexisearch Algorithm for the ATSP

Introducing a preprocessing technique as done in lexisearch algorithms of Pandit and Srinivas [9] and Ahmed [10] are not worthwhile for our algorithm. Of course, exchanging row with corresponding column in the cost matrix depending on their variances would have been worthwhile, but keeping record of track of the nodes would be more expensive. It is then observed that for some instances, solving transposed cost matrix takes lesser time with a same tour value than the original matrix. Now, the question is that under what condition the transposed matrix is to be considered instead of the given matrix. After studying many statistics of the cost matrix we come to the conclusion that when the variances (standard deviations) of rows are more than those of columns, our lexisearch algorithm takes less computational time. Hence, we introduce two preprocessing of the modified cost matrix before applying our simple lexisearch algorithm, as follows.

Let α_i and β_i be the standard deviations of i th row and i th column of the modified cost matrix C , for all $i=1, 2, \dots, n$.

Process 1: we count how many times $(\alpha_i < \beta_i)$, $\forall 1 \leq i \leq n$. If this count is greater than n , then the transposed modified cost matrix is considered.

Process 2: let $\alpha = \sum_{i=1}^n \alpha_i$ and $\beta = \sum_{i=1}^n \beta_i$. Next, we check whether $(\alpha < \beta)$. If yes, then the transposed modified cost matrix is considered.

The above preprocessing is incorporated in data-guided lexisearch algorithm. The data-guided lexisearch algorithm replaces the Steps 1 and 12 of the algorithm described in Section 3.4 as follows.

Step 0. Remove "bias" of the given cost matrix. Preprocess the modified cost matrix as described above and construct the "alphabet table" based on the modified cost matrix. Set "best solution value" as large as possible. Since "node 1" is the starting node, we start our computation from 1st row of the "alphabet table". Initialize "partial tour value" = 0, $r = 1$ and go to Step 2.

Step 11. Current word gives the optimal tour sequence with respect to the cost matrix used for solution. If the transposed matrix is used for solution, then take reverse of the tour sequence as the optimal tour sequence with respect to the original cost matrix and then stop.

5. Computational Experience

The simple lexisearch algorithm (LSA) and data-guided lexisearch algorithm (DGLSA) have been encoded in Visual C++ on a Pentium IV personal computer with speed 3GHz and 448 MB RAM under MS Windows XP operating system. We have selected asymmetric TSPLIB [25] instances of size from 17 to 71. We report the solutions that are obtained within four hours on the machine as well as total computational times (TotTime) in seconds. We also report percentage of error of the solutions obtained by the algorithms. The percentage of error is given by the formula $\text{Error}(\%) = (\text{BestSol} - \text{OptSol}) / \text{OptSol} \times 100\%$, where BestSol

Table 5: Results by data-guided lexisearch algorithm using Process 1 and Process 2.

Instance	OptSol	Process 1			Process 2		
		BestSol	Error(%)	TotTime	BestSol	Error(%)	TotTime
br17	39	39	0.00	80.56	39	0.00	80.56
ftv33	1286	1286	0.00	742.48	1286	0.00	161.84
ftv35	1473	1473	0.00	379.42	1473	0.00	379.42
ftv38	1530	1530	0.00	1868.85	1530	0.00	1868.85
p43	5620	5634	0.25	14400.00	5634	0.25	14400.00
ftv44	1613	1613	0.00	11334.98	1613	0.00	11334.98
ftv47	1776	1853	4.34	14400.00	1853	4.34	14400.00
ry48p	14420	16381	13.60	14400.00	16381	13.6	14400.00
ft53	6905	8116	17.54	14400.00	8116	17.54	14400.00
ftv55	1608	1711	6.41	14400.00	1711	6.41	14400.00
ftv64	1839	2052	11.58	14400.00	2052	11.58	14400.00
ft70	38673	40529	4.80	14400.00	40529	4.80	14400.00
ftv70	1950	2108	8.10	14400.00	2108	8.10	14400.00
Average			5.12	9969.71		5.12	9925.05

denotes the best/optimal solution obtained by the algorithms and OptSol denotes the optimal solution reported in TSPLIB.

Table 5 presents the comparative study of DGLSA using two different preprocessing methods, Process 1 and Process 2. It is seen from the table that the solutions obtained by both methods are same. Also, except for the instance ftv33, computational times are same. For ftv33, Process 2 is found to be better than Process 1. Hence, we consider DGLSA using Process 2 for comparison with other algorithms.

For the purpose of performance comparison of our algorithms, the implementation of lexisearch algorithm of Pandit and Srinivas [9], we named as PSA, reported in Srinivas [21], is encoded in Visual C++ and run on same machine and tested on same data set. The results are shown in Table 6. Out of thirteen instances only four, five, and five instances were solved optimally within four hours by PSA, LSA, and DGLSA, respectively. For the remaining instance, except for three instances—ry48p, ftv64, and ftv70, LSA finds better solution than PSA. On the other hand, except for two instances, ftv47 and ft53, DGLSA finds better solution than LSA. On average, PSA, LSA, and DGLSA find solutions which are 7.70%, 5.84%, and 5.12% away from the exact optimal solutions. This tells us that among the three algorithms our data-guided algorithm is the best.

Table 6 shows that, on the basis of total computational time, except for br17 and ftv35, LSA takes less time than PSA to solve the instances. On the other hand, DGLSA takes time that is less than or equal to the time taken by LSA for all instances. Table 6 also reports the computational time when the final solution is seen for the first time (FirstTime). In fact, a lexisearch algorithm first finds a solution and then proves the optimality of that solution, that is, all the remaining subproblems are discarded. The table shows that, on average computational time, PSA, LSA, and DGLSA find a solution within at most 50%, 39%, and 30% of the total computational time, respectively. That is, PSA, LSA, and DGLSA spend at least 50%, 61%, and 70% of total computational time on proving the solutions. Therefore, for these TSPLIB instances, PSA spends a relatively large amount of time on finding solution compared to LSA, and LSA spends a relatively large amount of time on

Table 6: Results by different algorithms on thirteen asymmetric TSPLIB instances.

Instance	OptSol	PSA				LSA				DGLSA			
		BestSol	Error(%)	FirstTime	TotTime	BestSol	Error(%)	FirstTime	TotTime	BestSol	Error(%)	FirstTime	TotTime
br17	39	39	0.00	0.22	46.27	39	0.00	9.20	80.56	39	0.00	9.20	80.56
ftv33	1286	1286	0.00	1040.23	1091.33	1286	0.00	46.25	161.84	1286	0.00	46.25	161.84
ftv35	1473	1473	0.00	385.98	787.73	1473	0.00	62.95	1625.80	1473	0.00	65.73	379.42
ftv38	1530	1530	0.00	7835.70	8991.98	1530	0.00	295.30	2894.69	1530	0.00	444.97	1868.85
p43	5620	5670	0.89	1.41	14400.00	5636	0.28	108.61	14400.00	5634	0.25	780.66	14400.00
ftv44	1613	1780	10.35	14329.52	14400.00	1613	0.00	7601.94	11334.98	1613	0.00	7601.94	11334.98
ftv47	1776	1780	0.23	3427.92	14400.00	1780	0.23	8401.03	14400.00	1853	4.34	9717.16	14400.00
ry48p	14420	15841	9.85	3057.05	14400.00	16381	13.60	6066.56	14400.00	16381	13.60	6066.56	14400.00
ft53	6905	8946	29.56	6850.77	14400.00	7728	11.92	4417.14	14400.00	8116	17.54	256.25	14400.00
ftv55	1608	1804	12.19	13902.94	14400.00	1735	7.90	25.78	14400.00	1711	6.41	2151.92	14400.00
ftv64	1839	2097	14.03	6966.64	14400.00	2178	18.43	11220.49	14400.00	2052	11.58	1556.53	14400.00
ft70	38673	42783	10.63	4764.20	14400.00	40640	5.09	12431.34	14400.00	40529	4.80	2.53	14400.00
ftv70	1950	2190	12.31	8039.19	14400.00	2309	18.41	331.31	14400.00	2108	8.10	9865.09	14400.00
Average			7.70	5430.91	10809.02		5.84	3924.45	10099.84		5.12	2966.52	9925.05

Table 7: Results by different algorithms on six asymmetric TSPLIB instances.

Ins.	DGLSA			ATSP6			SST2		
	BestSol	Error(%)	TotTime	BestSol	Error(%)	TotTime	BestSol	Error(%)	TotTime
br17	39	0.00	80.56	39	0.00	10.88	—	—	—
ftv33	1286	0.00	161.84	1286	0.00	5540.85	1286	0.00	668.00
ftv35	1473	0.00	379.42	1463.41	0.66	19271.70	1463.41	0.66	1357.00
ftv38	1530	0.00	1868.85	1488.22	2.81	8953.23	1520.19	0.65	1741.00
p43	5634	0.25	14400.00	5611.00	0.34	35193.60	—	—	—
ftv44	1613	0.00	11334.98	1594.94	1.13	159105.00	—	—	—
Average		0.04	4704.28		0.82	38012.54		0.44	1255.33

Table 8: Computational times on random asymmetric instances.

n	PSA		LSA		DGLSA	
	Average	Std dev	Average	Std dev	Average	Std dev
30	1.49	2.53	0.86	1.70	1.06	1.78
35	14.76	30.24	12.54	21.83	12.12	23.14
40	365.33	920.45	168.81	248.34	104.66	211.70
45	780.03	1518.12	945.32	1357.14	667.34	1069.07
50	1819.72	4354.40	2275.09	3201.82	1123.40	1904.31
Average	491.94	1830.94	543.26	1499.83	316.72	924.83

finding solution compared to DGLSA. So, a small number of subproblems are thrown by PSA in comparison to our algorithms. On the other hand, DGLSA requires largest amount of time on proving the solutions, and hence, a large number of subproblems are thrown. On the basis of computational time also, LSA is found to be better than PSA, and DGLSA is found to be the best one. There is good improvement of DGLSA over LSA for the instances in terms of solution quality and computational time. So, our goal is achieved very well.

Many recent good approximation methods based on branch and bound have been reported in the literature, but the nature of those algorithms are not same as our algorithms. Hence we cannot compare our algorithms with those algorithms. However, the performance of DGLSA is compared to the performance of integer programming formulation by Sherali et al. [11] (called ATSP6 therein) that was run on Dell Workstation PWS 650 with double 2.5GHz CPU Xeon processors, 1.5GB RAM under Windows XP operating system and implemented using AMPL (version 8.1) along with CPLEX MIP Solver (version 9.0). The same formulation was also implemented by Öncan et al. [14] (called SST2 therein) on a Pentium IV PC with 3GHz CPU using barrier solver of CPLEX 9.0. Unfortunately, a direct comparison on the same computer cannot be made, since, to the best of our knowledge, no online source code is available. However, our machine is around 1.5 times faster than the machine used for ATSP6 and is approximately same as machine used for SST2 (see machine speed comparison from Johnson [26]). Table 7 reports results for six asymmetric TSPLIB instances by DGLSA, and computational times (in seconds), solutions (lower bounds), and percentage of error of ATSP6 and SST2. The percentage of error is given by the formula $\text{Error}(\%) = (\text{OptSol} - \text{BestSol}) / \text{BestSol} \times 100\%$, where BestSol denotes the best solution (lower bound) obtained by ATSP6 and SST2.

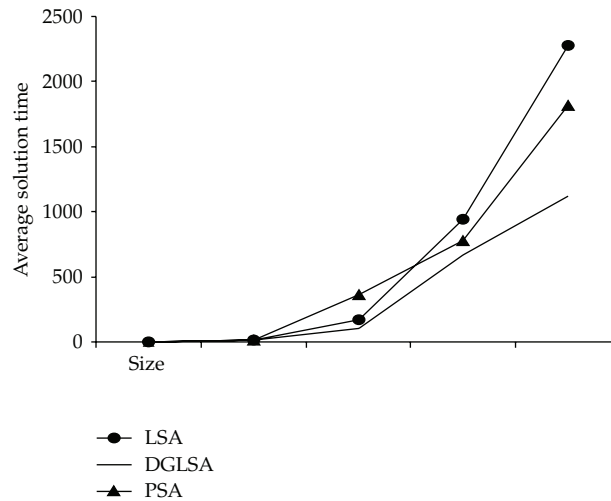


Figure 1: Trend of average computational (solution) times for different algorithms.

From Table 7, it is seen that for br17, DGLSA takes more computational time than ATSP6. For the remaining instances DGLSA takes least computational time. Out of six instances, two and five instances were solved optimally by DGLSA and ATSP6 respectively. On average, DGLSA is the best which finds solution that is 0.04% away from the exact optimal solution. As a whole, DGLSA is found to be best. Also, solutions by lexisearch algorithms do not rely on commercial math software.

We also run the lexisearch algorithms for asymmetric randomly generated instances of different sizes. We generate 20 different instances for each size. We include two statistics to summarize the results: average computational time (in seconds) and standard deviation of times. Table 8 presents results for asymmetric instances drawn from uniform distribution of integers in the interval $[1, n]$.

On the basis of average computational times, Table 8 shows that for $n = 30$, LSA is the best and PSA is the worst, and there is no improvement in average computational time and time variations by DGLSA over LSA, of course, DGLSA is better than PSA. For $n = (35$ and $40)$, LSA is better than PSA, and there is some improvement in average times by DGLSA over LSA, and hence, DGLSA is the best. For $n = (45$ and $50)$, PSA is better than LSA, and there is very good improvement in average times by DGLSA over LSA, and DGLSA is found to be the best. Overall there is an improvement of more than 41% on average computational time by DGLSA over LSA. Of course, for few problems DGLSA takes more times than LSA. We investigated in this direction, but we could not come to any conclusion in this regard. To visualize more clearly the trends of different algorithms as size of the problem increases, we present, in Figure 1, curves of average computational (solution) times by the algorithms against increasing sizes. It clear from Figure 1 that as the size increases PSA is performing better than LSA, and DGLSA is performing best.

On the basis of time deviations, Table 8 shows that PSA is the worst and DGLSA is the best. PSA shows large variations in the context of computational time for different instances of same size, and the problems seem to fall into two distinct groups with a big “gap” between the two groups, which is similar to the observation made by Ahmed [22]. As the size increases the variations amongst the computational times increase rapidly. LSA shows lesser variances

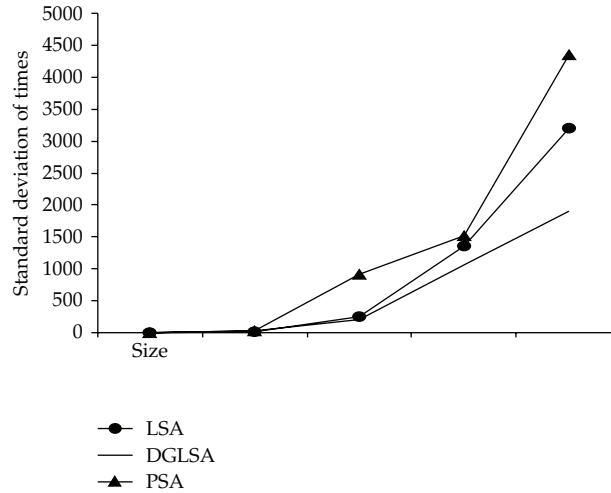


Figure 2: Trend of time variances for different algorithms.

than PSA, and DGLSA shows lowest variances. In fact, one of objectives of LSA was to reduce the gap amongst the computational times over PSA, which is achieved to some extent, and there is an improvement of more than 18% on the overall time variance by LSA over PSA. To reduce more in time variances, DGLSA is developed, and there is an improvement of around 38% and 50% on the overall time variance over LSA and PSA, respectively. We also present, in Figure 2, curves of standard deviations amongst computational times by the algorithms against increasing sizes. It clear from Figure 2 that as size increases, time variances by PSA are increasing rapidly than by LSA and DGLSA, and the trend of increment is lowest for DGLSA. Of course, for small sized problems variation by DGLSA is high.

To prove further that performance of DGLSA is the best for any asymmetric instance, we consider the instances drawn from uniform distribution of integers in different intervals. Table 9 presents the average computational times and standard deviations of the times for the values of n from 30 to 45 for the instances drawn from uniform distribution of integers in the intervals $[1, 10]$, $[1, n]$, $[1, 10^2]$, $[1, 10^3]$, and $[1, 10^4]$. The performance of the DGLSA does not change very much when the cost ranges increase from $[1, 10]$ to $[1, 10^4]$. On the basis of the average computational times, Table 9 shows that LSA is better than PSA, and DGLSA is the best for the instances drawn in the intervals $[1, 10]$ and $[1, n]$; but for the instances drawn in the intervals $[1, 10^2]$ and $[1, 10^3]$, DGLSA is better than LSA, and PSA is the best. For the instances drawn in the interval $[1, 10^4]$, there is less difference among the performance of different algorithms, and DGLSA is the best. However, on the basis of overall average computational times and variations in times, PSA is better than LSA, and DGLSA is the best. Asymmetric TSPLIB instances can be considered "hard", that is, even small instances take large computational times and take more times than for asymmetric random instances.

Further, to show that LSA and DGLSA are better than PSA, we run twenty symmetric TSPLIB instances of size from 14 to 76 and report computational times and solutions within four hours of computational times in Table 10. Since the condition for taking transpose matrix is not satisfied for symmetric instances, the results obtained by LSA and DGLSA are same, and hence, we report results by LSA only. In fact applying DGLSA makes no sense for symmetric instances because there is no any computational benefit in solving transposed

Table 9: Computational times for random asymmetric instances drawn from different intervals of data.

Interval	n	No. of Instances	PSA		LSA		DGLSA	
			Average	Std dev	Average	Std dev	Average	Std dev
[1, 10]	30–45	80	577.91	1860.81	338.93	1319.01	281.34	899.65
[1, n]	30–45	80	287.66	921.44	276.83	772.14	192.62	594.60
[1, 10^2]	30–45	80	214.05	1153.17	461.72	2278.94	220.18	627.33
[1, 10^3]	30–45	80	68.04	151.58	147.83	312.02	112.30	333.00
[1, 10^4]	30–45	80	434.08	2382.00	459.14	123.89	420.41	1911.07
Average			314.59	1497.05	339.29	1515.41	244.66	1021.90

Table 10: Results by the lexisearch algorithms for some symmetric TSPLIB instances.

Instance	OptSol	PSA				LSA			
		BestSol	Error(%)	FirstTime	TotTime	BestSol	Error(%)	FirstTime	TotTime
burma14	3323	3323	0.00	0.00	0.03	3323	0.00	0.00	0.01
ulysses16	6859	6859	0.00	0.16	1.59	6859	0.00	0.23	0.84
gr17	2085	2085	0.00	1.36	3.50	2085	0.00	0.36	0.81
gr21	2707	2707	0.00	1.41	2.16	2707	0.00	0.01	0.05
ulysses22	7013	7013	0.00	155.00	952.61	7013	0.00	96.86	271.04
gr24	1272	1272	0.00	4.47	34.55	1272	0.00	33.59	54.31
fri26	937	937	0.00	301.41	552.73	937	0.00	33.14	96.97
bayg29	1610	1610	0.00	101.20	168.97	1610	0.00	107.33	158.93
bays29	2020	2020	0.00	584.89	773.35	2020	0.00	0.23	245.23
dantzig42	699	734	5.01	4472.33	14400.00	699	0.00	11973.78	14400.00
swiss42	1273	1377	8.17	1523.67	14400.00	1360	6.83	10762.53	14400.00
att48	10628	12224	15.02	1063.81	14400.00	10850	2.09	10.42	14400.00
gr48	5046	5067	0.42	4127.78	14400.00	5525	9.49	11863.67	14400.00
hk48	11461	13222	15.37	9659.94	14400.00	12506	9.12	8969.51	14400.00
eil51	426	486	14.08	5500.09	14400.00	430	0.94	4812.47	14400.00
berlin52	7542	9177	21.68	9249.66	14400.00	8240	9.25	2256.58	14400.00
brazil58	25395	29433	15.90	1090.66	14400.00	29935	17.88	136.36	14400.00
st70	675	827	22.52	496.98	14400.00	748	10.81	5906.02	14400.00
eil76	538	698	29.74	7067.08	14400.00	633	17.66	1893.67	14400.00
pr76	108159	131056	21.17	2915.11	14400.00	118909	9.94	30.97	14400.00
Average			8.45	2415.85	8044.47		4.70	2944.39	7961.41

cost matrix. It is to be noted that lexisearch algorithms for the problem do not require any modification for solving different type of instances. Out of twenty instances, nine and ten instances were solved optimally by PSA and LSA, respectively. For the remaining instances, except for two instances—gr48 and brazil58, LSA finds better solution than PSA. On average, PSA and LSA find optimal solution 8.45% and 4.70% away from the exact optimal solutions, respectively. Regarding computational time, except for two instances gr24 and gr48, LSA

finds better final solutions quicker than PSA. On average, PSA and LSA find final solution within 31% and 37% of the total computational times, respectively. That is, PSA and LSA spend 69% and 63% of total time on establishing final solutions. From this computational experience, we can conclude that for the TSPLIB as well as random instances, DGLSA is the best among all algorithms considered in this study, and it finds optimal solution quickly for asymmetric instances.

6. Conclusion

We presented simple and data-guided lexiseach algorithms that use path representation method for representing a tour for the benchmark asymmetric traveling salesman problem to obtain exact optimal solution to the problem. The performance of our algorithms is compared against lexiseach algorithm of Pandit and Srinivas [9] and solutions obtained by integer programming formulation of Sherali et al. [11] for some TSPLIB instances. It is found that our data-guided algorithm is very effective for these instances. For random asymmetric instances also, the data-guided lexiseach algorithm is found to be effective, and the algorithm is not much sensitive to the changes in the range of the uniform distribution.

For symmetric TSPLIB instances our simple algorithm is found to be better than the algorithm of Pandit and Srinivas [9]. However, the proposed data-guided algorithm is not applicable to symmetric instances, and hence, we do not report any computational experience for random symmetric instances. In general, a lexiseach algorithm first finds an optimal solution and then proves the optimality of that solution. The lexiseach algorithm of Pandit and Srinivas [9] spends a relatively large amount of time on finding optimal/best solution compared to our algorithms.

We have investigated that using our proposed algorithms asymmetric TSPLIB instances could be solved of maximum size 45, whereas asymmetric random instances could be solved of maximum size 50 within four hours of computational time. This shows that asymmetric TSPLIB instances are harder than asymmetric random instances. For symmetric instances, our data-guided algorithm is not applicable, hence we could not draw any conclusion on these instances. However, we could only find optimal solution and could not prove optimality of the solution, using our simple algorithm of maximum size 42. Of course, we did no modification in the algorithm for applying on different types of instances.

Though we have proposed a data-guided module of the lexiseach algorithm, still for some instances it takes more times than the simple lexiseach algorithm, and it is applicable only for the asymmetric instances. So a closer look at the structure of the instances and then developing a more sophisticated data-guided module may apply on symmetric instances, and may further reduce the computational time and provide better solutions for large instances. Another direction of the research is to propose a better lower bound technique which may reduce the solution space to be searched, and hence reduce the computational time.

Acknowledgments

The author wishes to acknowledge Professor S. N. Narahari Pandit, Hyderabad, India for his valuable suggestions and moral support. This research was supported by Deanery of Academic Research, Al-Imam Muhammad Ibn Saud Islamic University, Saudi Arabia via Grant no. 280904. The author is also thankful to the anonymous honorable reviewer for his valuable comments and suggestions.

References

- [1] C. P. Ravikumar, "Solving large-scale travelling salesperson problems on parallel machines," *Microprocessors and Microsystems*, vol. 16, no. 3, pp. 149–158, 1992.
- [2] R. G. Bland and D. F. Shallcross, "Large traveling salesman problems arising from experiments in x-ray crystallography: a preliminary report on computation," *Operations Research Letters*, vol. 8, no. 3, pp. 125–128, 1989.
- [3] C. H. Papadimitriou and K. Steglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall of India Private Limited, New Delhi, India, 1997.
- [4] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," *Operations Research Letter 2*, vol. 2, pp. 161–163, 1983.
- [5] D. Naddef, "Polyhedral theory and branch-and-cut-algorithms for the symmetric TSP," in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. P. Punnen, Eds., vol. 12 of *Computational Optimization*, pp. 29–116, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
- [6] M. Fischetti, A. Lodi, and P. Toth, "Exact methods for the asymmetric traveling salesman problem," in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. P. Punnen, Eds., vol. 12 of *Computational Optimization*, pp. 169–205, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
- [7] Z. H. Ahmed, "A lexisearch algorithm for the bottleneck traveling salesman problem," *International Journal of Computer Science and Security*, vol. 3, no. 6, pp. 569–577, 2010.
- [8] S. N. N. Pandit, "The loading problem," *Operations Research*, vol. 10, no. 5, pp. 639–646, 1962.
- [9] S. N. N. Pandit and K. Srinivas, "A lexisearch algorithm for the traveling salesman problem," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 3, pp. 2521–2527, November 1991.
- [10] Z. H. Ahmed, "A data-guided lexisearch algorithm for the bottleneck traveling salesman problem," *International Journal of Operational Research*. In press.
- [11] H. D. Sherali, S. C. Sarin, and Pei-F Tsai, "A class of lifted path and flow-based formulations for the asymmetric traveling salesman problem with and without precedence constraints," *Discrete Optimization*, vol. 3, no. 1, pp. 20–32, 2006.
- [12] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a large-scale traveling-salesman problem," *Operational Research Society Journal*, vol. 2, pp. 393–410, 1954.
- [13] S. C. Sarin, H. D. Sherali, and A. Bhootra, "New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints," *Operations Research Letters*, vol. 33, no. 1, pp. 62–70, 2005.
- [14] T. Öncan, İ. K. Altinel, and G. Laporte, "A comparative analysis of several asymmetric traveling salesman problem formulations," *Computers & Operations Research*, vol. 36, no. 3, pp. 637–654, 2009.
- [15] E. Balas and P. Toth, "Branch and bound methods," in *The Traveling Salesman Problem*, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan et al., Eds., Wiley Series in Discrete Mathematics & Optimization, pp. 361–401, John Wiley & Sons, Chichester, UK, 1985.
- [16] G. Carpaneto, M. Dell'Amico, and P. Toth, "Exact solution of large-scale, asymmetric traveling salesman problems," *Association for Computing Machinery. Transactions on Mathematical Software*, vol. 21, no. 4, pp. 394–409, 1995.
- [17] D. L. Miller and J. H. Pekny, "Exact solution of large asymmetric traveling salesman problems," *Science*, vol. 251, no. 4995, pp. 754–761, 1991.
- [18] M. Turkensteen, D. Ghosh, B. Goldengorin, and G. Sierksma, "Tolerance-based branch and bound algorithms for the ATSP," *European Journal of Operational Research*, vol. 189, no. 3, pp. 775–788, 2008.
- [19] S. N. N. Pandit, "An Intelligent approach to travelling salesman problem," Symposium in Operations Research, Khragpur: Indian Institute of Technology (1964).
- [20] M. S. Murthy, *Some Combinatorial Search Problems (A Pattern Recognition Approach)*, Ph.D. thesis, Kakatiya University, Warangal, India, 1979.
- [21] K. Srinivas, *Data Guided Algorithms in Optimization and Pattern Recognition*, Ph.D. thesis, University of Hyderabad, Hyderabad, India, 1989.
- [22] Z. H. Ahmed, *A Sequential Constructive Sampling and Related Approaches to Combinatorial Optimization*, Ph.D. thesis, Tezpur University, Assam, India, 2000.

- [23] Z. H. Ahmed and S. N. N. Pandit, "The travelling salesman problem with precedence constraints," *Opsearch*, vol. 38, no. 3, pp. 299–318, 2001.
- [24] S. N. N. Pandit, S. C. Jain, and R. Misra, "Optimal machine allocation," *Journal of Institute of Engineers*, vol. 44, pp. 226–240, 1964.
- [25] TSPLIB, <http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>.
- [26] D. S. Johnson, Machine comparison site, <http://public.research.att.com/~dsj/chtsp/speeds.html>.



Hindawi

Submit your manuscripts at
<http://www.hindawi.com>

