ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A DATA GUIDED LEXI-SERACH APPROACH FOR TIME DEPENDENT TRAVELLING SALEMSMAN PROBLEM

Dr.K.Sobhan Babu*
* Department of Mathematics, JNTUK-UCEV, Vizianagaram, A.P., INDIA.

Abstract

Abstract. A simple lexi-search algorithm that uses path representation method for the time dependent traveling salesman problem TDTSP is proposed, to obtain exact optimal solution to the problem. Then a data-guided lexi search algorithm is presented. First, the cost matrix of the problem is transposed depending on the variance of rows and columns, and then the simple lexi-search algorithm is applied. It is shown that this minor preprocessing of the data before the simple lexi-search 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.

Keywords: Time Dependent Travelling Salesman Problem, Lexi-Search, Data Guided Lexi-Search, Pattern Recognition

Introduction

The travelling salesman problem (TSP) finds application in a variety of situations such as automatic drilling of printed circuit boards and threading of scan cells in testable VLSI circuit [1], X-ray crystallography [2], and so forth. On the basis of structure of the cost matrix, the TSPs are classified in to two groups-symmetric (STSP) and asymmetric (ATSP). The TSP is symmetric if $c_{ij} = c_{ji}, \forall\, i, j$ 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. Some ATSP instances are more complex, in many cases; ATSP instances are transformed into STSP instances and subsequently solved using STSP algorithms [4].

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], lexi-search [7, 8] are well known exact algorithms. In our investigation, we apply lexi-search algorithm to obtain exact optimal solution to the problem. The lexi-search algorithm has been successfully applied to many combinatorial optimization problems. Pandit and Srinivas [9] showed that lexi-search algorithm is better than the branch and bound algorithm. In lexi-search algorithm, lower bound plays a vital role in reducing search space, hence, reduces computational time. Also, preprocessing of data, before the lexi-search algorithm is applied, can reduce computational effort substantially [9, 10].

In this paper, we first present a simple lexi-search algorithm using path representation for a tour for obtaining exact optimal solution to the TDTSP. Then a data-guided lexi-search 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 lexi-search 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. Section 3 represents the mathematical formulation. A simple lexi-search algorithm is developed in Section 4. Section 5 presents a data-guided lexi-search algorithm for the problem. Computational results for the algorithms and conclusions has been presented in Section 6.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

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, O¨ ncan 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 lexi-search 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 lexisearch algorithm of Pandit [19], which is found to be better than previous lexisearch 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 lexi-search 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.

Mathematical Formulation

$$\boldsymbol{Minimize\ Z} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \boldsymbol{C(i,j,k)\ X(i,j,k)}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} X(i,j,k) = \delta_k\,, \qquad \delta_k = 0\ or\ 1; \ \sum_{k=1}^{n}\delta_k = m_0$$

$$\sum_{j=1}^{n}\sum_{k=1}^{n} X(i,j,k) = \delta_i\,, \qquad \delta_i = 0\ or\ 1; \ \sum_{i=1}^{n}\delta_i = m_0$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} X(i,j,k) = \delta_j\,, \qquad \delta_j = 0\ or\ 1; \ \sum_{j=1}^{n}\delta_j = m_0$$

Where δ_i =1 indicates that the salesman started from city i, otherwise 0. δ_j =1 indicates that the salesman visited from city j, otherwise 0. δ_k =1 indicates the time/facility for the tour form city i to city j, otherwise 0. In addition to the above constraints X will be a feasible solution as follows. If it gives a tour for the salesman who visits any of the m_0 cities in m_0 times/facilities with the condition that a point of time in his tour he will not visit more than one pair of cities. The problem is to find a tour for set of m_0 cities out of n cities such that the total cost of the tour of m_0 cities is minimum.

Lexi-Search Algorithm

In lexi-search 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].

Feasibility Criterion of a partial word

A feasibility criterion is developed, in order to check the feasibility of a partial word $L_{k+1} = (a_1, a_2. . .a_k, a_{k+1})$ given that, L_k is a partial word. Let IR be an array, IR (i) =1, i ε N represents that the salesman is visiting some city from city i otherwise 0. IC be an array where IC (i) =1, i ε N represents that the salesman is coming to city i from some city, otherwise 0. IT be an array where IT (i) =1, ii ε N represents that the salesman at time i travels one pair of cities. SW be an array where SW (i) is the city that the salesman visiting from city i and SW (i) = 0 indicates that the salesman is not visiting any city from city i. ISC be an array where ISC (i) is the number of times the index i as a city or time involved the word L_k. Then for a given partial word $L_k = (a_1, a_2. . .a_k, a_k)$ the values of the arrays IR,IC,IT,SW,ISC are as follows.

$IR(R(a_i)) = 1$, $i = 1,2,\dots,k$ and $IR(j) = 0$ for other elements of j.

$IC(C(a_i))=1$, $i=1,2,\dots,k$ and $IC(j) = 0$ for other elements of j.

$IT(T(a_i))=1$, $i=1,2,\dots,k$ and $IT(j) = 0$ for other elements of j.

$SW(R(a_i) = C(a_i)$, $i=1, 2\dots, k$ and SW (j) for other values of j.

$$\left.\begin{aligned} ISC(R(a_i)) &= ISC(R(a_i)) + 1 \\ ISC(C(a_i)) &= ISC(C(a_i)) + 1 \\ ISC(T(a_i)) &= ISC(T(a_i)) + 1 \end{aligned}\right| \quad i = 1, 2\dots k$$

The recursive algorithm for checking the feasibility of a partial word is as follows. In the algorithm first we equate

IX=0.

$TR = R(a_{p+1})$; $TC = C(a_{p+1})$; $TT = T(a_{p+1})$;

```
STEP 1: IX=0; TCX=TC; IDXT=IDX;      GOTO 2.

STEP 2: IS (IR (TR) =1)   IF YES GOTO 12; IF NO GOTO 3.

STEP 3: IS(IC (TC) =1)    IF YES GOTO 12; IF NO GOTO 4.

STEP 4: IS (IT (TT) =1)   IF YES GOTO 12; IF NO GOTO 5.

STEP 5: IS (ISC (TR).EQ.0)      IF YES IDXT = IDXT + 1 GOTO 6; IF NO GOTO 6.

STEP 6: IS (ISC (TC).EQ.0)      IF YES IDXT = IDXT + 1 GOTO 7; IF NO GOTO 7.

STEP 7: IS (ISC (TT).EQ.0)      IF YES IDXT = IDXT + 1 GOTO 8; IF NO GOTO 8.

STEP 8: IS (IDXT.GT.M)          IF YES GOTO 12; IF NO GOTO 9.

STEP 9: IS (SW (TCX) =0)        IF YES IX=1 GOTO 12; IF NO IK=SW (TCX) GOTO 10.

STEP 10: IS (IK=TR)             IF YES GOTO 11; IF NO TCX=IK GOTO 9.

STEP 11: IS (I=N)               IF YES IX=1 GOTO 12; GOTO 12.

STEP 12:        STOP.
```

At the end if IX=1 then the partial word is feasible, other wise it is infeasible. This recursive algorithm is used as a subroutine in the Lexi-Search algorithm(vide 4.6), to check the feasibility of a partial word. Search starts with a very large value (= 9999999) as the trail value of VT. If the value of a feasible word is known, it can as well be taken as the value of VT. During the search VT is improved (in fact, it gets decreased). At the end of the search, the current value VT gives value of the optimal feasible word. A partial word L_k, is constructed as $L_k = L_{k-1}*(a_k)$. (Where * indicates concatenation) and V (L_k) and LB (L_k) are calculated. Then two situations arise: one for branching and the other for continuing search.

[1].LB(L_k) ≥ VT: if it is the case, we reject the partial word, i.e., the block of words with L_k as leader is rejected for not having an optimal word and we also reject all the partial words of order K the succeed L_k.

[2].LB (L_k) < VT: if it is so, we check whether L_k is feasible. If it is feasible, we proceed to consider a partial word a order which represents a sub-block of words represented by L_k. If L_k is not feasible, we consider the next partial word of order k, by considering another letter in kth position which succeeds a_k. If all the partial words of order k are exhausted, then we consider the next partial word of order (k-1).

Data Guided Lexi-Search Algorithm

Introducing a preprocessing technique as done in lexi search algorithms of Pandit and Srinivas and Ahmed 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 lexi search algorithm takes less computational time. Hence, we introduce two preprocessing of the modified cost matrix before applying our simple lexi search algorithm as follows:

Let α_i, β_i be the standard deviations of ith row and jthcolumn of the modified cost matrix C, for all i = 1,2, …,n.

Process 1: count how many time ($\alpha_i < \beta_i$), for all $1 \le i \le n$. 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 $\beta = \sum_{j=1}^{n} \beta_j$.

Next, we check whether ($\alpha < \beta$). If yes, then the transposed
modified cost matrix is considered.

The above preprocessing is incorporated in data-guided lexi search algorithm. The data-guided lexi search algorithm replaces the steps 1 and 12 of the algorithm described in feasibility criterion of Lexi-search algorithm.

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.

Computational results & Conclusion

The lexi-search algorithm (LSA) and data-guided lexi-search algorithm (DGLSA) have been encoded in C on a Pentium IV personal computer with speed 3GHz and 448 MB RAM under MS windows XP operating system. We have selected TSPLIB [22] instances of size from 17 to 55. We report the solutions that are obtained within three 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 Error(%) = (BestSol-

OptSol)/OptSol×100%, where BestSol denotes the best/optimal solution obtained by the algorithms and OptSol denotes the optimal solution reported in TSPLIB.

Table 1: Results by data-guided lexi-search algorithm using Process 1 and Process 2

Instance	Opt Sol	BestSol	Process 1		BestSol	Process 2	
			Error (%)	TotTime		Error (%)	TotTime
br17	39	39	0.00	75.23	39	0.00	75.23
ftv33	1286	1286	0.00	644.22	1286	0.00	611.26
ftv35	1473	1473	0.00	354.12	1473	0.00	345.88
ftv38	1530	1548	0.00	1649.29	1548	0.00	1649.29
p43	5620	5631	0.14	13566.10	5631	0.14	12866.10
ftv44	1613	1613	0.00	11323.24	1613	0.00	12899.28
ftv47	1776	1811	2.13	13566.28	1811	2.13	12866.10
ry48p	14420	15344	10.21	13566.28	15344	10.21	13222.11
ftv53	6905	7925	15.11	13228.01	7925	15.11	13222.01
ftv55	1608	1710	5.35	13566.28	1710	5.35	13561.10
Average			**3.29**	**8153.90**		**3.29**	**8131.83**

Table 1 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. Hence, we consider DGLSA using Process 2 for comparison with other algorithms.

Table 2 shows that, DGLSA takes time that is less than or equal to the time taken by LSA for all instances. The table shows that, on average computational time, LSA, and DGLSA find a solution within at most 39%, and 30% of the total computational time, respectively. That is, LSA, and DGLSA spend at least 61%, and 70% of total computational time on proving the solutions. Therefore, for these TSPLIB instances, LSA spends a relatively large amount of time on finding solution compared to DGLSA.

Table 2: Results by different algorithms on ten TSPLIB instances

Instance	Opt Sol	LSA			DGLSA		
		BestSol	Error (%)	TotTime	BestSol	Error (%)	TotTime
br17	39	39	0.00	75.23	39	0.00	75.23
ftv33	1286	1286	0.12	611.24	1286	0.00	611.26
ftv35	1473	1473	1.02	345.12	1473	0.00	345.88
ftv38	1530	1548	1.23	1649.29	1548	0.00	1649.29
p43	5620	5631	1.35	13566.10	5631	0.14	12866.10
ftv44	1613	1613	1.36	11323.24	1613	0.00	12899.28
ftv47	1776	1811	3.37	13566.28	1811	2.13	12866.10
ry48p	14420	15344	10.21	13566.28	15344	10.21	13222.11

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

ftv53	6905	7925	15.11	13228.01	7925	15.11	13222.01
ftv55	1608	1710	5.35	13566.28	1710	5.35	13561.10
Average			**3.91**	**8153.90**		**3.29**	**8131.83**

On the other hand, DGLSA requires largest amount of time on proving the solutions, and hence, a large number of sub problems are thrown. On the basis of computational time also, LSA is found to be better, 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. We presented simple and data-guided lexi-search 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. It is found that our data-guided algorithm is very effective for these instances. For random instances also, the data-guided lexi-search algorithm is found to be effective, and the algorithm is not much sensitive to the changes in the range of the uniform distribution.
Though we have proposed a data-guided module of the lexi-search algorithm, still for some instances it takes more times than the simple lexi-search 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.

Acknowledgements

The author whishes to acknowledge Prof.S.N.N Pandit (late) and acknowledge Prof. Sundara Murthy.M for their continuous help and valuble suggensions and moral support to strengthern of the paper.The author is also thankful to the anonymous honorable reviewer for his valuable comments and suggestions.

References

[1] C.P.Ravi kumar, “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, Dodrecht, 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, Dodrecht, 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] Sobhan Babu.K., Chandra Kala.K, Purushotam. S, , Sundara Murthy.M., “A New Approach for

variant multi Assignment Problem" International Jouranal on Computer Science and Engineering, Vol.2, No.5, 2010, pp.1634-1641.

[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. O¨ ncan, I¨. 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, JohnWiley & 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] S. N. N. Pandit, "An Intelligent approach to travelling salesman problem," Symposium in Operations Research, Khragpur: Indian Institute of Technology, 1964.

[18] Z. H. Ahmed, "A Data-Guided Lexisearch Algorithm for the Asymmetric Traveling Salesman Problem," *Mathematical Problems in Engineering*, Vol.2011, doi:10.1155/2011/750968.

[19] Z. H. Ahmed and S. N. N. Pandit, "The travelling salesman problem with precedence constraints," *Opsearch*, vol. 38, no. 3, pp. 299–318, 2001.

[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]TSPLIB,http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

[23] D. S. Johnson, Machine comparison site, http://public.research.att.com/∼dsj/chtsp/speeds.html.