ISSN No. 0976-5697

Volume 3, No. 1, Jan-Feb 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

An Exact Algorithm for Travelling Salesman Problem

Dr. K.Sobhan Babu*
Assistant Professor in Mathematics
U C E K, J N T U Kakinada
KAKINADA, A.P., India
sobhanjntu@gmail.com

Dr. Keshava Reddi.E
Associate Professor in Mathematics
College of Engineering, J N T U Anantapur
Anantapur, A.P., India
eddualakr@yahoo.co.in

Prof.Sundara Murthy.M
Senior Professor in Mathematics
SRI Venkateswara University, Tirupati
Chittor, A.P., India
profmurthy@gmail.com

Abstract: A well known example of a Combinatorial Optimization Problem is the Travelling Salesman Problem. The Combinatorial Programming Problems share the following properties: They are Optimization Problems, are easy to state, and have a finite but usually very large number of feasible solutions. Lexi-Search is by far the mostly used tool for solving large scale NP-hard Combinatorial Optimization problems. Lexi-Search is, however, an algorithm paradigm, which has to be filled out for each specific problem type, and numerous choices for each of the components exist. Even then, principles for the design of efficient Lexi-Search algorithms have emerged over the years. Although Lexi-Search methods are among the most widely used techniques for solving hard problems, it is still a challenge to make these methods smarter. The motivation of the calculation of the lower bounds is based on ideas frequently used in solving problems. Computationally, the algorithm extended the size of problem and find better solution.

Keywords: Travelling Salesman Problem, Tour, Lexi-Search, Word, Pattern

I. INTRODUCTION

The Travelling Salesman Problem (TSP) is a classical problem of combinatorial optimization of operations research area. The purpose is to find a shortest tour through a given no. of locations such that every location is visited exactly once. The cost of travelling from location i to location j is denoted by C_{ij}. These costs are symmetric if $C_{ij} = C_{ji}$ for each of pair of cities i and j, and asymmetric otherwise. There are several practical uses for this problem, such as vehicle routing with the additional constraints of vehicle's route, such as capacity of vehicles [1], drilling problems [2], minimize waste [3], clustering data arrays [4], X-ray crystallography [5], shot sequence generation for scan lithography [6] and many others.

This problem has also been used during the last years as comparison basis for improving several optimizations techniques, such as genetic algorithms [7], simulated annealing [8], tabu search [9], local search [10], ant colony [11] and Branch and Bound (B&B). The principal types of B&B used to solve the TSP are: The best known exact algorithms are based on either the B&B method for the Asymmetric TSP (ATSP) [12] or the Branch and Cut (B&C) method for the Symmetric TSP (STSP) using the double index formulation of the problem [13]. Currently, most algorithms for the TSP ignore high cost arcs or edges and save the low cost ones. A drawback of this strategy is that costs of arcs and edges are not accurate indicators whether those arcs or edges are saved in an optimal TSP solution.

State-of-the-art B&B algorithms for the ATSP can be found in [14]&[15], [16]&[17] and later [18]. The algorithms by Miller and Carpaneto apply patching to obtain upper bounds, use AP lower bounds, and branch on a smallest cycle in the current AP solution, i.e., a cycle of smallest cardinality. The search strategy of both algorithms is BFS. This means that for many ATSP instances, solutions are obtained in very short time. On the other hand, a list of sub problems should be maintained in order to determine the most promising one. As a consequence, BFS algorithms tend to run out of memory when the search trees grow large. Upper bounds of B&B algorithm for the ATSP was improved by [18] with iterative patching which is a procedure for constructing good feasible solutions of the ATSP. Iterative patching procedures reduce the number of sub problems in the spanning B&B tree, so that smaller computation times are needed. This method only considers the time consuming problems.

In this paper, while claiming that our algorithm is faster than that of Branch and Bound for most of the problems, fresh computational results which are obtained after refining the arrangement of the alphabet table for the Truncated Time Dependent Travelling Salesman Problem (TTDTSP) is being presented in this paper. In Section 2, a Lexi-Search method was developed for the TTDTSP and mathematical formulation is shown in Section 3. The algorithm is presented in Section 4. The computational results are provided in Section 5 and the concluding remarks are given in Section 6.

© 2010, IJARCS All Rights Reserved

II. AN ALGORITHM FOR TTDTSP

The name ***Lexicographic-search*** or ***Lexi-search*** method implies that the search is made for an optimal solution in a systematic way, just as one search for meaning of a word in a dictionary. When the process of feasibility checking of a partial word becomes difficult, though lower bound computation is easy, ***Pattern Recognition Technique*** [19] can be used. Lexi-Search algorithms, in general, require less memory, due to the existence of Lexicographic order of partial words. If Pattern Recognition Technique is used, the dimension requirement of the problem can be reduced, since it reduces to the two-dimensional cost array into a linear and the problem can be reduced to a linear form of finding an optimal word of length n [19] and hence reduces computational work in getting an optimal solution.

III. MATHEMATICAL FORMULATION

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

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

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

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

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.

IV. THE ALGORITHM

The name ***Lexicographic-search*** or ***Lexi-search*** method implies that the search is made for an optimal solution in a systematic way, just as one search for meaning of a word in a dictionary. When the process of feasibility checking of a partial word becomes difficult, though lower bound computation is easy, ***Pattern Recognition Technique*** [19] can be used. Lexi-Search algorithms, in general, require less memory, due to the existence of Lexicographic order of partial words. If Pattern Recognition Technique is used, the dimension requirement of the problem can be reduced, since it reduces to the two-dimensional cost array into a linear and the problem can be reduced to a linear form of finding an optimal word of length n [19] and hence reduces computational work in getting an optimal solution. The concepts and notations involved in the Lexi-Search are briefly described below.

A. Pattern Recognition Technique:

The search efficiency of a Lexi-Search algorithm is based on the choice of an appropriate Alphabet-Table. In this case two conflicting characteristics of the search list have to be taken into account: one is the difficulty in setting bounds to the values of the partial words (that defines partial solutions representing subsets of solutions). The other difficulty is in checking the feasibility of a partial word. Thus two cases arises in the choice of Alphabet Table [19][20].

(i). The process of checking the feasibility of a partial word is easy, while the calculations of a lower bound is bulky and

(ii). Computation of lower bound is easy, while the feasibility checking is difficult.

When the process of feasibility checking of a partial word becomes difficult and the lower bound computation is easy, a modified Lexi-Search i.e. Lexi-Search with recognizing the Pattern of the solution known as Pattern Recognition Technique [19][20] can be adopted. In this method, in order to improve the efficiency of the algorithm, first the bounds are calculated and then the partial word, for which the value is less than the initial (trial) value are checked for the feasibility.

B. Pattern:

An indicator matrix X, associated with an appropriate assignment of tasks to the agents is defined as a *Pattern*. A Pattern is said to be **feasible,** if X is feasible. Each pattern X can also be represented by the set of all ordered triples {(i, j, k)}, for which X (i, j, k) =1. In general, there will be m*n*k ordered pairs in a matrix X (m, n, k).

C. Alphabet Table & Word:

Let SN = (1,2, . . . ,n^3) be the set of indices, BD be an array of corresponding costs of the ordered pairs and DD be the array of cumulative sums of elements in BD. Let arrays R, C and T be respectively row, column and time/facility indices of the ordered triples. Let $L_k = (a_1,a_2. . .a_k)$. $a_i \varepsilon$ SN be a ordered sequence of k indices from S. The pattern represented by the ordered triples indices are given by L_k is independent of the order of a_i in the sequence. For uniqueness, the indices in L_k are arranged in increasing order, such that $a_i < a_{i+1}$, i = 1,2, . . .,k-1. The set S is defined as ***Alphabet-Table*** with alphabetic order as (1,2, . . .,n^3) and the ordered sequence L_k is defined as a word of length k. A word L_k is said to be *sensible* word if $a_i < a_{i+1}$, i = 1, 2. . . k-1; *non sensible* otherwise. It is said to be feasible, if it represents a ***feasible*** pattern. Any of the letters in S can occupy the first place in a word L_k. Our interest is only in set of words of length atmost equal to n, since the words of length greater than n are necessarily infeasible, as any feasible pattern can have only n unit entries in it. If k < n, L_k is called a ***Partial word*** and if K = n, it is a full length word or simply a word. A partial word L_k represents a block of words with L_k as a leader i.e. as its first k letters. A leader is

© 2010, IJARCS All Rights Reserved

said to be feasible, if the block of words defined by it has at least one feasible word.

D. Value of the Word:

The value of the (partial) word L_k, $V(L_k)$ is recursively defined as $V(L_k) = V(L_{k-1}) + BD(a_k)$ with $V(L_0) = 0$, where BD is the cost array arranged such that, $BD(a_k) < BD(a_{k+1})$. $V(L_k)$ and the value of the pattern X, will be the same, since X is the (partial) pattern represented by L_k.

E. Lowerbound of a Partial Word:

A lower bound $LB(L_k)$, for the values of the blocks of words represented by $L_k = (a_1, a_2 \ldots a_k)$ can be defined as follows:

$$LB(L_k) = V(L_k) + \sum_{j=1}^{m_0-k} V(a_k + j) = V(L_k) + VC(a_k + m_0 - k) - VC(a_k)$$

It can be seen that $LB(L_k)$ is the value of the complete word, which is obtained by concatenating the first $m_0 - k$ letters to the partial word L_k.

F. Feasibility criterion of a Partial Word:

A feasibility criterion is developed, in order to check the feasibility of a partial word $L_{k+1} = (a_1, a_2 \ldots a_k, a_{k+1})$ 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 $L_k = (a_1, a_2 \ldots a_k, a_k)$ the values of the arrays IR,IC,IT,SW,ISC are as follows.

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

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

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

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

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

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, otherwise 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.

(i) $LB(L_k) \geq VT$: 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.

(ii) $LB(L_k) < VT$: 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 k^{th} 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).

V. COMPUTATIONAL RESULTS

A Computer program for the proposed algorithm is written in C language and is tested on the COMPAQ system. We tried a set of problems for different sizes. Random numbers are used to construct the Time matrix. The following table gives the list of the problems tried along with the average CPU time in seconds required for solving them.

© 2010, IJARCS All Rights Reserved

Table: 1(CPU run time in seconds for Lexi-Search Algorithm)

Sr. No.	*Problem Dimensions*			*AT*	*Type-I*			*Type-II*			*Type-III*		
	m	n	l		Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
1	5	6	2	0.0000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	8	10	3	0.0004	0.001	0.004	0.0025	0.004	0.005	0.0045	0.003	0.001	0.002
3	10	10	3	0.04	0.0041	0.0006	0.0047	0.0059	0.0214	0.0136	0.0047	0.0153	0.0100
4	10	25	3	0.09	0.660	0.943	0.8016	0.893	0.941	0.917	0.968	0.1	0.534
5	30	50	3	2.146	2.141	2.562	2.3515	2.468	2.9	2.684	2.61	2.8	2.705

VI. CONCLUSIONS

The Lexi - Search algorithm presented in this chapter, incorporating Pattern Recognition Technique is tested. The same problem sets have been tested with using C language and successfully applied to many real problems. The findings of the model-testing and a wide range of sensitivity analyses using an artificially generated data set are presented. Both solution procedures prove to be efficient and effective in providing close to optimal solutions and proven with a surprisingly small number of patterns. Lexi-search algorithms are proved to be more efficient in many combinatorial problems. Lexi Search strategy stands as a good candidate for being an AI (Artificial Intelligence) search mechanism. Apart from the minimal requirement of memory, the Lexi Search helps to obtain optimality, faster than Branch and Bound in many cases. Further, Lexi Search clearly specifies Simpler rules for Branching, Bounding and Termination. Even with the restrictions imposed, the Lexi – Search takes reasonably less time. Its efficiency over the Bounding Procedures (like Lagrangean Relaxation) in Branch & Bound method is also significant. Further it is observed that with the modification of the sort procedure while arranging the alphabet table, the Lexi Search algorithm is becoming more efficient. On the whole, it is felt that Lexi Search algorithm is faster than the Branch and Bound algorithm.

VII. REFERENCES

[1]. Laporte, G.," The vehicle rotting problem: an overview of exact and approximate algorithms", Eur. J. Oper. Res. 59 (2), (1992) 345–358.

[2]. Onwubolu, G.C. and Clerc, M.," Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization", Int.J. Prod. Res. 42 (3), (2004) 473–491.

[3]. Grafinkel, R. S "Minimizing wallpaper waste, part I: a class of Traveling Salesman Problems", Oper Res 25, (1977) 741-751.

[4]. McCormick, W. T., Schweitaer, P. J. and White, T. W."Problem decomposition and data reorganization by a Clustering technique", Oper Res 20, (1972) 993-1009.

[5]. Bland, R. G.and Shallcross, D.F., "Large Traveling Salesman Problem arising experiment in x-ray

[6]. Crystallography: A Preliminary reported on computation", Oper Res 8, (1989) 125-128.

[7]. Shinano, Y.,Inui, N., Fukagawa, Y. and Takakura, N., "An Application of Traveling- Salesman Models to Shot Sequence Generation for Scan Lithography", 5th European Congress on Computational Methods in Applied Sciences and Engineering, (2008) 30 –Julys 5, Venice, Italy.

[8]. Affenzeller, M., Wanger, S., "A self-adaptive model for selective pressure handling within the theory of genetic algorithms", EUROCAST 2003, Las Palmas de Gran Canaria, Spain, Lect. Notes Comp. Sci. 2809 (1),(2003) 384–393.

[9]. Budinich, M., 1996," A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing", Neural Comput. 8, pp. 416–424.

[10]. Liu, G., He, Y., Fang, Y., Oiu, Y.," A novel adaptive search strategy of intensification and diversification in tabu search", in: Proceedings of Neural Networks and Signal Processing, Nanjing, China.(2003)

[11]. Bianchi, L., Knowles, J., Bowler, J.," Local search for the probabilistic traveling salesman problem: Correction to the 2-p-opt and 1-shift algorithms", Eur. J. of Oper. Res. 162(1), (2005) 206–219.

[12]. Chu, S.C., Roddick, J.F., Pan, J.S.," Ant colony system with communication strategies", Inform. Sci. 167 (1–4), (2004), 63– 76.

[13]. Fischetti, M., Lodi, A., Toth, P., Exact Methods for the Asymmetric Traveling Salesman Problem. In: Gutin, G., Punnen, A.P. (Eds.), The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht, (2002). 169–194 (Chapter 9).

[14]. Naddef, D., 2002, Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP.In: Gutin, G., Punnen, A.P. (Eds.), The Traveling Salesman Problem and its Variations. Kluwer Dordrecht, (1991) 29–116 (Chapter 2).

[15]. Miller, D. and Pekny, J.,"Exact Solution of Large Asymmetric Traveling Salesman Problems", Science, 251: 754–761.

[16]. Pekney, J.F., Miller, D.L.,"Exact solution of the no-wait flow shop scheduling problem with a comparison to heuristic methods", Computers & Chemical Engineering (1991) 15:741-748.

[17]. Carpaneto, G., Dell'Amico, M. and Toth, P., "Exact Solution of Large-scale Asymmetric Traveling Salesman Problems", ACM Transactions on Mathematical Software, 21,(1995) 394–409.

[18]. Carpaneto, G., Toth, P.," Some new branching and bounding criteria for the asymmetric traveling salesman problem", Management Science 21, (1980) 736–743.

[19]. Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G., "Tolerance-based Branch and Bound algorithms for the ATSP", European Journal of Operational Research 189: (2007) 775–788, Available online at www.sciencedirect.com.

[20]. Sundara Murthy, M. "Combinatorial Programming: A Pattern Recognition Approach", A Ph.D., Thesis, REC, Warangal. (1979).

[21]. Sobhan Babu,K & Sundara Murthy.M, "An efficient algorithm for Variant Bulk Transportation Problem", International Journal of Engineering Science and Technology, 2010, Vol.2(7), pp.2595-2600.

© 2010, IJARCS All Rights Reserved

Short Bio Data for the Authors


Dr.K.SOBHAN BABU is presently working a Assistant Professor in the department of Mathematics, UCEK, JNTUK and Additional Controller of Examinations in JNTUKAKINADA.


Dr.E.KESHAVA REDDI is presently working a Associate Professor in the department of Mathematics, College of Engineering, Anantapur and Controller of Examinations in JNTU Anantapur. His Research area includes Optimization, Data Mining etc.


Dr.M.SUNDARA MURTHY is a senior Professor in the Department of Mathematics, Sri Venkateswara University, Tirupati. He has so many publications in National and International reputed Journals.

© 2010, IJARCS All Rights Reserved