AJCSE
ASIAN JOURNAL OF COMPUTER SCIENCE ENGINEERING

www.ajcse.info

Asian Journal of Computer Science Engineering 2017; 2(3): 17-24

RESEARCH ARTICLE

A Lexi Search Approach to Generalized Travelling Salesman Problem

***Dr. U. Balakrishna[1]**

**Professor of Mathematics, Department of Science & Humanities, Sreenivasa Institute of Technology and Management Studies, Chittoor*

Received on: 25/03/2017, Revised on: 02/04/2017, Accepted on: 25/05/2017

ABSTRACT

There are n cities and N = {1, 2, 3… n}. The cost matrix C(i, j) [i, j=1,2,3,…..,n] is the cost of the salesman visiting from city i to city j is given. We are given a partition of cities into groups. The restriction for the salesman is that he should visit from one group to another group. The problem is to find a feasible tour for n cities for the salesman with minimum total cost with the above restriction. We propose a Lexi Search Algorithm based on Pattern Recognition Technique for solving "Generalized Traveling Salesman Problem" with an illustrative example.

For this problem a computer program is developed for the algorithm and is tested. It is observed that it takes less time for solving fairly higher dimension problem also.

Keywords: Generalized Travelling Salesman Problem (GTSP), Lexi Search algorithm, Pattern Recognition Technique, Trip-schedule, Pattern, Alphabet-table, Word.

INTRODUCTION:

The Generalized Travelling Salesman Problem was first addressed in [8, 19]. Exact algorithms can be found in Laporte et al. [10, 11], Laporte & Nobert [12], Fischetti et al. [5, 6], and other in [2, 14]. On the other hand, several worthy heuristic approaches have been applied to solve the Generalized Travelling Salesman Problem. Noon [13] presented several heuristics for the Generalized Travelling Salesman Problem among which the most promising one was an adaptation of the well-known nearest-neighbor heuristic for the Travelling Salesman Problem. Similar adaptations of the farthest-insertion, nearest-insertion, and cheapest-insertion heuristic were proposed by Fischetti et al [5]. GI^3 (Generalized Initialization, Insertion, and Improvement) is one of the most sophisticated heuristics; which was developed by Renaud and Boctor [17]. GI^3 Is a generalization of the I^3 heuristic presented in Renaud et al [18]. Other exact algorithms are presented in [3, 4, 9, 20 and 22]. But in all the above attempts the simple combinatorial structure of the GTSP is not at all taken into consideration. Several researchers [1, 15, 16 and 20] have implemented Lexi Search approach for the standard TSP, with mixed results. The Lexi Search approach is found new best solutions for some well-studied benchmark problems. The standard TSP is called 'Two Dimensional TSP'. The standard TSP has been generalized in many directions. In the present paper

1) We have introduced some practical constraint i.e., cluster constraint of the problem
2) We have implementing the Lexicographic Search Approach
3) We have made use of Pattern Recognition Technique which takes care of the simple combinatorial structure of the problem.

There exist several applications of the GTSP such as

1) Postal routing
2) Computer files processing
3) Order picking in warehouses
4) Process planning for rotational parts
5) The routing of clients through welfare agencies.

Furthermore, many other combinatorial optimization problems can be reduced to the GTSP problem. The GTSP is NP-hard since it is a special case of the TSP which is partitioned into m clusters with each containing only one node.

Here we study the following "Generalized Travelling Salesman Problem" (GTSP):

"There are n cities and N= {1, 2, 3… n}. The cost array C (i, j) is the cost of a salesman visiting from city i to city j (i, j=1, 2, 3… n)".The set of n cities are divided into r groups or clusters suchthat

***Corresponding Author:** U. Balakrishna **, Email:** prasanti_balu@rediffmail.com

$N=\{N_1, N_2, N_3, ..., N_r\}$ and $N_i \cap N_j = \varnothing$ .The restriction is the Traveling Salesman has to visit from one cluster to another cluster.

The problem is to find a minimum cost tour by visiting n cities, with the above restriction.

MATHEMATICAL FORMULATION:

The solution X (i, j), $(i, j) \in I\ X\ J$ be defined as:

X (i, j) = 1, if the salesman visits city j from city i = 0, otherwise

and $N=N_1 \cup N_2 \cup N_3 \cup \ldots\ldots \cup N_r$, $N_i \cap N_j=\Phi$, $i \neq j$

Then the problem can be defined as:

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n} C(i, j)\, X(i, j) \qquad (1)$$

Subject to:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} X(i, j) = 1 \qquad (2)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} X(i, j) = n \qquad (3)$$

$$X(i, j) = 0 \text{ or } 1,\ (i, j) \in IxJ \qquad (4)$$

Where i,j={1,2,3,…..,n} are the sets of cities and C(i,j) is the amount of cost by visiting i^{th} city to j^{th} city. It is to be noted that equations (2) to (4) define the constraint set of the generalized traveling salesman problem, whose objective function is $\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n} C(i, j)\, X(i, j)$.

X=X (i, j) is a feasible tour if it satisfies all the above conditions.

Numerical illustration:

The concepts and the algorithm developed will be illustrated by a numerical example for which n=6, let N= [1, 2, 3, 4, 5, 6].

ENCODING:

1.Group coding: First two cities i.e., 1,2 are considered as first group; next two cities i.e., 3,4 are considered as second group; and the remaining cities 5,6 are considered as third group; i.e.,

	1	2	3	4	5	6
GN	1	1	2	2	3	3

Where GN (i) = j indicates that the i^{th} city in the j^{th} cluster or group

Table –1

$$C(i,j) = \begin{bmatrix} 9999 & 14 & 27 & 2 & 10 & 26 \\ 17 & 9999 & 15 & 22 & 4 & 8 \\ 22 & 17 & 9999 & 16 & 71 & 54 \\ 1 & 7 & 17 & 9999 & 5 & 29 \\ 51 & 31 & 41 & 5 & 9999 & 21 \\ 61 & 71 & 14 & 1 & 7 & 9999 \end{bmatrix}$$

In the numerical example given in Table-1, C (i, j), i=j, i, j=1,2,3,4,5,6 are taken to be very high number 9999 because they are irrelevant for finding tours of the salesman. Though the entire C (i, j) are taken as non – negative integers it can be easily seen that this is not a necessary condition and the cost as well be negative quantities.

CONCEPTS AND DEFINITIONS:

- **Lexi Search Method**

  The Lexi search derives its name from lexicography "The science of effective storage and retrieval of information". The solution space of the problem is arranged in a block-hierarchical order, like the words in a dictionary. Each node is considered as a letter in an alphabet and each tour can be represented as a word with this alphabet. The entire set of words in this dictionary is partitioned into "blocks", defined and identified by their "leader" i.e., the first few letters of the words in the block, which are common. By the structure of the problem, it is often possible to obtain bounds to the values of all the words in a block by an examination of its leader. Hence, by comparing that bound with the value of a trial solution, one can jump over to either the next "block" or next "super-block" if the current one cannot contain any word better than the trial solution. In case the block contains a "better word", one can go into the sub-block.

  Let a, b, c and d be the four nodes to be covered. The words starting with "a" can be grouped together. Such a group is called a "block" and node "a" is called a leader of the block. In a block there can be sub-blocks. For instance, block of words with leader "a" has sub-blocks {abc, abcd, abdc} with a leader"ab". There could be blocks with only one word, for instance, the block with leader "abd " has only one word. All incomplete words can be used as leaders to def'ine blocks. For each of the blocks with leaders "ab", "ac", "ad" the block with leader "a" is a "super-block".

It is noted that blocks with are mutually exclusive and collectively exhaustive can be constructed. For this problem, it is clarified that an incomplete (partial) word does not necessarily imply a sub tour and our "lexi-search" is for words without sub tours. The search is started with the first incomplete word "a" as the leader. As the search is for an optimum solution, we jump a block if its bound is greater than or even equal to the current trial solution: no complete word in the block can be better than the current one (through there may be a word that is equally good)

- **Definition Of Pattern**

  An indicator two-dimensional array which is associated with a tour is called a "pattern". A pattern is said to be feasible if X is a feasible solution. Now V(x) the value of the pattern X is defined as V (X) = $\sum_{i=1}^{n}\sum_{j=1}^{n} C(i,j).X(i,j)$.

  The value V(X) gives the total cost of the tour for the solution represented by X. Thus the value of the feasible pattern gives the total cost represented by it. In the algorithm, which is developed in the sequel, a search is made for a feasible pattern with the least value. Each pattern of the solution X is represented by the set of ordered triples [(i, j)] for which X (i, j)=1, with the understanding that the other X(i, j)'s are zeros.

- **Definition Of An Alphabet – Table And A Word**

  There are M= *n x n* ordered doubles in the two-dimensional array X. For convenience these are arranged in ascending order of their corresponding costs and are indexed from 1 to M (Sundara Murthy-1979). Let SN= [1, 2, 3…M] be the set of M indicies. Let D be the corresponding array of costs. If a, b∈SN and a<b then D (a) ≤ D (b). Also let the arrays R, C be the array of row, column Indicies of the ordered doubles represented by SN and DC be the array of cumulative sums of the elements of D. The set SN is defined as the "Alphabet-Table" with alphabetic order as (1, 2, 3… M).The arrays SN, D, DC, R, C for the numerical example are given in the table-2. If P ∈ SN then (R(P), C(P)) is the ordered double and D(a)= C(R(a),C(a)) is the value of the ordered triple and DC(a)= $\sum_{i=1}^{a} D(i)$

Table-2(Alphabet – Table)

S.	D	DC	R	C		S.	D	DC	R	C
1	1	1	4	1		**16**	17	143	3	2
2	1	2	6	4		**17**	17	160	4	3
3	2	4	1	4		**18**	21	181	5	6
4	4	8	2	5		**19**	22	203	2	4
5	5	13	4	5		**20**	22	225	3	1
6	5	18	5	4		**21**	26	251	1	6
7	7	25	4	2		**22**	27	278	1	3
8	7	32	6	5		**23**	29	307	4	6
9	8	40	2	6		**24**	31	338	5	2
10	10	50	1	5		**25**	41	379	5	3
11	14	64	1	2		**26**	51	430	5	1
12	14	78	6	3		**27**	54	484	3	6
13	15	93	2	3		**28**	61	545	1	6
14	16	109	3	4		**29**	71	616	3	5
15	17	126	2	1		**30**	71	687	6	2

Let L_K= {a_1,a_2,……,a_K}, $a_i \in$ SN be an ordered sequence of k indicies from SN. The pattern represented by the ordered double whose indicies are given by L_k is independent of the order of a_i in the sequence. Hence for uniqueness the indicies are arranged in the increasing order such that $a_i \leq a_{i+1}$, i=1,2… n-1. The set SN is defined as the "Alphabet-Table" with alphabetic order as (1,2,3…n^2) and the ordered sequence L_K is defined as a "word" of length k. A word L_k is called a "Sensible word". If $a_i < a_{i+1}$, for i=1,2,3…k-1 and if this condition is not met it is called a "insensible word". A word L_K is said to be feasible if the corresponding pattern X is feasible and same is with the case of infeasible and partial feasible. Therefore a partial feasible word is said to be feasible if k=n.

A partial word L_k is said to be feasible if the block of words represented by L_K has at least one feasible word or, equivalently the partial pattern represented by L_k should not have any inconsistency. Any of the letters in SN can occupy the first place in the partial word L_k. Consider L_{K-1} = $(a_1,a_2,\ldots\ldots\ldots,a_{k-1})$. The alphabet for the kth position is $SNa_{k-1}=(a_{k-1}+1,a_{k-1}+2\ldots n^2)$, where SN_P is defined as SN_P $=(p+1,p+2,\ldots\ldots,n^2)$. Thus for example consider a word with two letters as $(a_1,a_2)=(1,3)$. Then $SNa_2=SN_3=(4,5,6\ldots30)$ is the alphabet for the third position. We concentrate on the set of words of length at most m (for the numerical example it is 6). A leader L_k (k<n) is said to be feasible, if the block of words defined by it contains at least one feasible word or equivalently there should not be inconsistency in the partial pattern defined by the partial word. The value of the L_K , $V(L_K)$ is defined recursively as $V(L_K)=V(L_{K-1})+D(a_K)$ with $V(L_0)=0$ obviously this $V(L_K)$ and V(X) the values of the pattern X represented by L_k will be same (Sundara Murthy-1979). For example consider $L_3=(1,3,4)$. Then $V(L_3)$ will be $V(L_3)=1+2+4=7$

LOWER BOUND OF A PARTIAL WORD LB (L_K)

A lower bound LB (L_k) for the values of the block of words represented by L_k can be defined as follows: $LB(L_K) = V(L_K) + \sum_{j=1}^{n-k} D(a_k + j) = V(L_K) + DC(a_K + n-k) - DC(a_K)$

Consider the partial word $L_3 = (1, 2, 4)$

$V(L_3) = 1+1+4=6$

$LB(L_3) = V(L_3) + DC(a_3+6-3) - DC(a_3)$

$= 6 + DC(4+6-3) - DC(4)$

$= 6 + DC(7) - DC(4) = 6+25-8 = 23$

FEASIBILITY CRITERION OF A PARTIAL WORD:

A recursive algorithm is developed for checking 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 feasible partial word. We will introduce some more notations which will be useful in the sequel.

RI be an array where RI (i) =1, i ∈ N represents that the salesman is visiting some city from city i, otherwise zero.

CI be an array where CI (i) =1, i ∈ N represents that the salesman is coming to city i from some city; otherwise zero.

SW be an array where SW (i) is the city that the salesman is visiting from city i, Sw(i)= 0 indicates that the salesman is not visiting any city from city i.

L be an array where L (i) is the letter in the ith position of a word.

GN be an array where GN(i) is the city in ith grade,GN(i)=1 indicates that the salesman is in ith grade.

Then for a given partial word $L_K = (a_1,a_2,\ldots..,a_K)$ the values of the arrays RI, CI, SW, L, GNR and GNC as follows.

$RI(R(a_i)) =1$, i=1, 2, 3……..K

$CI(C(a_i)) =1$, i=1, 2, 3……..K

$SW(R(a_i)) =C(a_i)$, i=1, 2, 3……..K

$L(i) =a_i$, i=1, 2, 3……..K

$GNR(R(a_i))=GN(R(a_i))$

$GNC(C(a_i))=GN(C(a_i))$

For example consider a sensible partial word L_3= (1, 2, 9) which is feasible. The array RI, CI, SW, L, GNR and GNC takes the values represented in the table-3 given below.

Table-3

	1	2	3	4	5	6
RI	0	1	0	1	0	1
CI	1	0	0	1	0	1
SW	0	6	0	1	0	4
L	1	2	9	-	-	-
GNR	0	1	0	2	0	3
GNC	1	0	0	2	0	3

The recursive algorithm for checking the feasibility of a partial word L_P is given as follows: In the algorithm first we equate IX=0. At the end if IX=1 then the partial word is feasible, otherwise it is infeasible. For this algorithm we have RI, CI, SW, TR=R (a_{p+1}), TC=C (a_{P+1}).

Algorithm-1:

STEP1 : IX=0
TCX=TC
GOTO 2

STEP 2 : IS (RI (TR) = 1) IF YES GOTO 8
IF NO GOTO 3

STEP 3 : IS (CI (TC) = 1) IF YES GOTO 8

```
                                IF NO GOTO 4

STEP4  : IS GS (TR)) = GS (TC) IF YES GOTO STEP 8
                                IF NO GOTO 5

STEP5  : IS (SW (TCX) = 0)     IF YES IX=1 GOTO 8
                          IF NO IK=SW (TCX) GOTO 6

STEP6  : IS (IK=TR)        IF YES GOTO 7
                           IF NO TCX=IK GOTO 5

STEP7  : IS (I=N)          IF YES IX=1 GOTO 8
                           IF NO GOTO 8

STEP8  : STOP
```

This recursive algorithm will be used as a subroutine in the lexi-search algorithm. We start the algorithm with a very large value, say, 9999 as a trial value of VT. If the value of a feasible word is known, we can as well start with that value as VT. During the search the value of VT is improved. At the end of the search the current value of VT gives the optimal feasible word. We start with the partial word $L_1 = (a_1) = (1)$. A partial word $L_p = L_{p-1} * (a_p)$ where $*$ indicates chain form or concatenation. We will calculate the values of V (L_p) and LB (L_p) simultaneously. Then two cases arises (one for branching and other for continuing the search).

1. LB (L_p) < VT. Then we check whether L_p is feasible or not. If it is feasible we proceed to consider a partial word of order (p+1), which represents a sub block of the block of words represented by L_p. If Lp is not feasible then consider the next partial word of order p by taking another letter which succeeds a_p in the p^{th} position. If all the words of order p are exhausted then we consider the next partial word of order (p-1).
2. LB (L_P) ≥ VT. In this case we reject the partial word meaning that the block of words with L_p as leader is rejected for not having an optimal word and we also reject all partial words of order p that succeeds L_p.

Now we are in a position to develop lexi search algorithm to find an optimal feasible word.

Algorithm 2: (Lexi-Search Algorithm)

The following algorithm gives an optimal feasible word.

```
STEP 1 : (Initialization) The arrays SN, D, DC, R, C and values of N, GN are made available RI, CI, SW, L, GNR,GNC, V, LB are initialized to zero. The values I=1, J=0, VT=9999, NZ=N * N * N –N, MAX=NZ-1
STEP 2 : J=J+1  IS (J>MAX)      IF YES GOTO 11
                                IF NO GOTO 3
STEP 3 : L (I) = J JA = J + N – I;
                        IS (I = 1)
                        IF YES  V (I) = D (J) GOTO 3B
                        IF NO GOTO 3A
STEP 3A: V (I) = V (I -1) + D (J)
                        GOTO 3B
STEP 3B:  LB (I) = V (I) + DC (JA) – DC (J)
                        GOTO 4
STEP 4 : IS (LB (I) ≥ VT) IF YES GOTO 11
                                IF NO GOTO 5
STEP 5 : TR=R (J)
                        TC=C (J)
                        GOTO 6
STEP 6 : CHECK THE FEASIBILITY OF L (USING ALGORITHM-1)
          IS (IX=0) IF YES GOTO 2
                    IF NO GOTO 7

STEP 7 : IS (I=N) IF YES GOTO 10
                  IF NO GOTO 8

STEP 8 : L (I) = J
                  RI (TR) = 1
                  CI (TC) = 1
                  SW (TR) = TC
                  GOTO 9
STEP 9 : I=I+1
                  MAX=MAX+1
                  GOTO 2
STEP10: L (I) =J
 L (I) IS FULL LENGTH WORD AND IS FEASIBLE.
          VT=V (I), record L (I), VT,
                  GOTO 13
STEP11: IS (I=1) IF YES GOTO 14
                  IF NO GOTO 12
STEP12: I=I-1
                  MAX=MAX+1
                   GOTO 13
STEP13: J=L (I)
                  TR = R (J)
                 TC = C (J)
         RI (TR) = 0
                 CI (TC) = 0
                 SW (TR) = 0
                 GOTO 2
STEP14: STOP
                 END
```

The current value of VT at the end of the search is the value of the optimal feasible word. At the end if VT = 9999 it indicates that there is no feasible solution.

Search table:

The working details of getting an optimal word, using the above algorithm for the illustrative

numerical example are given in the Table-4. The columns (1), (2), (3) and (4) gives the letters in the first, second, third and fourth places respectively. The corresponding V (I) and L B (I) are indicated in the next two columns. The row R, C and T gives the row, column and time indices of the letter. The last column gives the remarks regarding the acceptability of the partial words. In the following table A indicates ACCEPT and R indicates REJECT.

Table- 4: Search table

S.N	1	2	3	4	5	6	V	LB	R	C	REM
1	1						1	18	4	1	A
2		2					2	18	6	4	A
3			3				4	18	1	4	R
4			4				6	23	2	5	A
5				5			11	23	4	5	R
6				6			11	25	5	4	R
7				7			13	28	4	2	R
8				8			13	31	6	5	R
9				9			14	38	2	5	R
10				10			16	44	1	5	R
11				11			20	49	1	2	R
12				12			20	51	6	3	R
13				13			21	54	2	3	R
14				14			22	56	3	4	R
15				15			23	57	2	1	R
16				16			23	61	3	2	A
17					17		40	61	4	3	R
18					18		42	62	5	6	A
19						19	64	64	2	4	R
20						20	64	64	3	1	R
21						21	68	68	1	6	R
22						22	69	69	1	3	A=V T=69
23					19		45	67	2	4	R
24					20		45	71	3	1	R>VT
25				17			23	66	4	3	R
26				18			27	71	5	6	R>VT
27			5				7	26	4	5	R
28			6				7	29	5	4	R
29			7				9	34	4	2	R
30			8				9	41	6	5	R
31			9				10	48	2	6	A
32				10			20	48	1	5	A
33					11		34	48	1	2	R
34					12		34	49	6	3	R
35					13		35	51	2	3	R
36					14		36	53	3	4	R
37					15		37	54	2	1	R
38					16		37	54	2	3	A
39						17	54	54	4	3	R
40						18	58	58	5	6	R
41						19	59	59	2	4	R
42						20	59	59	3	1	R
43						21	63	63	1	6	R
44						22	64	64	1	3	R
45						23	66	66	4	6	R
46						24	68	68	5	2	R
47						25	78	78	5	3	R>VT
48					17		37	58	4	3	R
49					18		41	63	5	6	R
50					19		42	64	2	4	R
51					20		42	68	3	1	R
52					21		46	73	1	6	R>VT
53				11			24	53	1	2	R
54				12			24	55	6	3	R
55				13			25	58	2	3	R
56				14			26	60	3	4	R
57				15			27	61	2	1	R
58				16			27	65	3	2	A
59					17		44	65	4	3	R
60					18		48	70	5	6	R>VT
61				17			27	70	4	3	R>VT
62			10				12	55	1	5	A
63				11			26	55	1	2	R
64				12			26	57	6	3	R
65				13			27	60	2	3	A
66					14		43	60	3	4	R
67					15		44	61	2	1	R
68					16		44	61	3	2	R
69					17		44	65	4	3	R
70					18		48	70	5	6	R>VT
71				14			28	62	3	4	R
72				15			29	63	2	1	R
73				16			29	67	3	2	A
74					17		46	67	4	3	R
75					18		50	72	5	6	R>VT
76				17			29	73	4	3	R>VT
77			11				16	61	1	2	A
78				12			30	61	6	3	R
79				13			31	64	2	3	A
80					14		47	64	3	4	R
81					15		48	65	2	1	R
82					16		48	65	3	2	R
83					17		48	69	4	3	R=VT
84				14			32	65	3	4	R
85				15			33	67	2	1	R
86				16			33	71	3	2	R>VT
87			12				16	62	6	3	R
88			13				17	67	2	3	A
89				14			33	67	3	4	R
90				15			34	68	2	1	R
91				16			34	72	3	2	R>VT
92			14				18	69	3	4	R=VT
93		3					3	24	1	4	R
94		4					5	29	2	5	A
95			5				10	29	4	5	R
96			6				10	32	5	4	A
97				7			17	32	4	2	R
98				8			17	35	6	5	R
99				9			18	42	2	6	R
100				10			20	48	1	5	R
101				11			24	53	1	2	R
102				12			24	55	6	3	A
103					13		39	55	2	3	R
104					14		40	57	3	4	R
105					15		41	58	2	1	R
106					16		41	58	3	2	A
107						17	58	58	4	3	R
108						18	62	62	5	6	R
109						19	63	63	2	4	R
110						20	63	63	3	1	R
111						21	67	67	1	6	R
112						22	68	68	1	3	R
113						23	70	70	4	6	R>VT

At the end of the search the current value of VT is 69 and it is the value of the optimal feasible word L_6 = (1, 2, 4,16, 18, 22). It is given in the 22nd row of the search table. The array RI, CI, TI, SW, L, GNR and GNC takes the values represented in the Table-5 given below. The Pattern represented by the above optimal feasible word is represented in the following table-6.

Table – 5

	1	2	3	4	5	6
RI	1	1	1	1	1	1
CI	1	1	1	1	1	1
SW	3	5	2	1	6	4
L	1	2	4	16	18	22
GNR	1	1	2	2	3	3
GNC	1	1	2	2	3	3

Table-6

$$X(i,j) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The tour represented by the above pattern is [(1,3), (3,2), (2,5), (5,6),(6,4),(4,1)], where, the salesman, from city 1 goes to city 3 (i.e., from group 1 to group 2) , from city 3 goes to city 2 (i.e., from group 2 to group 1) , from city 2 goes to city 5 (i.e., from group 2 to group 3) , from city 5 goes to city 6(i.e., from group 3 to group 3), from city 6 goes to city 4(i.e., from group 3 to group 2) and from city 4 goes to city 1(i.e., from group 2 to group 1).

It also can be represented by

$$1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 1$$

Computational experience:

A Computer program for the above algorithm is written in C language and is tested. Random numbers are used to construct the cost matrix. The following table-7 gives the list of the problems tried along with the average CPU time in seconds required for solving them. In the table AT represents the CPU time to construct the alphabet-table and ET represents the CPU time taken for the search of a feasible word. The time is represented in seconds. In the table-8 'n' is the number of cities.

Experiments are carried and by generating the three different classes of random data sets, where the three types of data sets are defined as follows:

Type 1: C (i, j) are uniformly random in [1,100]
Type 2: a) C (i, j) are uniformly random in [1,100]
b) VT=0.85VT
Type 3: a) C (i, j) are uniformly random in [1,100]
b)Max=(nxnxn)/3

And the results are tabulated in Table. For each type, four data sets are tested. It is seen that time required for the search (ET) of the optimal solution is fairly less.

Table-7

Problem dimensions	No. of prob's	AT	Total time taken(ET)								
			TYPE 1			TYPE 2			TYPE 3		
N			MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
20	4	0.10	5.24	5.98	5.50	4.98	5.53	5.17	4.71	5.38	5.05
160	4	0.27	7.52	7.86	7.72	6.31	6.87	6.53	6.20	6.76	6.51
260	4	0.62	8.41	8.92	8.71	8.09	8.51	8.36	7.35	7.70	7.49

In the above table it can be noticed that the average CPU time for n=260 in Type 1 is 8.71seconds and for Type 2 it is 8.36 seconds. The reduction in time is because the search for optimal solution is made around 0.85VT. But in second case also we are getting the same optimal solution. In Type 3 the search is in 1/3 of the alphabet table, so it takes less time and interestingly here also we are getting the same optimal solution for these problems.

ACKNOWLEDGEMENTS

The author expresses my deep sense of reverence and gratitude to the research supervisor **Prof. M. Sundara Murthy**, Department of Mathematics, S.V.U. College of Engineering, Tirupati for suggesting this problem for investigation. It is solely due to his immense interest, competence and exceptional guidance, critical analysis, transcendent and concrete suggestions enlightened discussions, which cumulatively are responsible for the successful execution of this work.

REFERENCES

1. Bhavani, V. and Sundara Murthy, M.(2005):Time-Dependent Traveling Salesman Problem OPSEARCH 42, PP. 199-227.
2. Vladimir Dimitrijevic, Milan Milosavljevic, Milan Markovic(1996):A Branch and Bound algorithm for solving a generalized traveling salesman problem, UNIV.BEOGRAD.PUBL.ELEKTROTEHN. FAK. Ser. Mat. 7, 31-35.
3. Henry-Labordere, A (1969) :The record balancing problem-A dynamic programming solution of a generalized traveling salesman problem. Revue

Francaise D Informatique De Recherche Operationnelle 3 (NB2), 43-49

4. Laporte, G, Asef-Vaziri, A and Sriskandararajah, C (1996): Some applications of the generalized traveling salesman problem. Journal of the Operational Research Society 47 (12) 1461-1467
5. Laporte, G, Mercure, H and Nobert, Y (1985):Finding the shortest Hamiltonian circuit through n clusters: A Lagrangian approach, Congressus Numerantium 48,277-290
6. Laporte, G, Mercure, H and Nobert, Y(1987):Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. Discrete Appl. Math, 18, 185-197
7. Laporte,GandNobert,Y.(1983):Generalized traveling salesman problem through n-sets of nodes-An integer programming approach, INFOR 21 (1) 61-75.
8. Pandit,S.N.N.andRajbhougshi(1976):Restricted TSP through n sets of nodes. Paper presented at the 9th Annual Convention of ORSI, Calcutta
9. Pandit, S.N.N. and Srinivas, K(1962):A Lexisearch algorithm for traveling Salesman problem, IEEE, 2521-2527.
10. Snyder, L.V and Daskin, M.S.(2006):A random-key genetic algorithm for the generalized traveling salesman problem Technical Report 04T-018, Dept. of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA.
11. Sundara Murthy, M(1979):Combinatorial Programming - A Pattern Recognition Approach. PhD, Thesis REC, Warangal, India