A PATTERN RECOGNITION LEXI SEARCH APPROACH TO TRAVELLING SALESMAN PROBLEM WITH ADDITIONAL CONSTRAINTS

Dr. K. CHENDRA SEKHAR

Lecturer, Sarvodaya Degree College, Nellore, AP, INDIA,
Email:deekshitha_balu@rediffmail.com

Dr.U.BALAKRISHNA

Professor, Sreenivasa Institute of Technology & Management studies, Chittor
Email:prasanti_balu@rediffmail.com

Dr. E. PURUSHOTHAM
Lecturer, SV Degree College, Tirupati, AP, INDIA

C.SURESH BABU
Research Schlor, SV University, Tirupati, AP, INDIA

Dr. M. SUNDARA MURTHY
Professor, SV University, Tirupati

Abstract - There are n cities and N = {1, 2,… n}. Let {1} be the headquarter city and the sub-headquarter cities i.e.,H = {a_1, a_2… a_h} be the subset of N. The cost array C (i, j) indicates the cost of the travelling salesman by visiting the j^{th} city from i^{th} city. Suppose the salesman wants to visiting the m (m<n) (| M | = m & M $\subset$ N, M = {1} U N_1 U {h} U N_2) cities with the condition that the person starts his trip schedule from a headquarter city (say1) visiting N_1 cities before reaching any one of the sub-headquarter city (say h, h $\in$ H) and he will come back to the home city by visiting N_2 cities m=N_1+N_2+2. The objective of the problem is that the total cost of the trip schedule of the salesman under the considerations should be least/minimum. The model can be expressed as a zero-one programming problem.

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

***Keywords**: TSPAC, Lexi search algorithm, Pattern recognition technique, Trip schedule, pattern, Alphabet-table, word*

I INTRODUCTION

The travelling salesman problem (TSP) is one of the most widely studied combinatorial programming problems in the literature of Operations Research. Many researchers have been developed different algorithms for the solution of TSP so far. Here, we shall present an integrated overview of some of the exact and approximate algorithms developed for the solution.

The methods considered usually can be divided into three basic parts: a starting point, a solution generation scheme, and a termination rule. When the termination rule is such that the iteration stops if and only if a tour is optimal, the method is exact, and when the rule is such that the iteration stops, if but not only if a tour is optimal, the method is approximate.

Unfortunately, there is no such analytical method which can be used satisfactorily. However, a good number of algorithms have been proposed for the travelling salesman problem either optimally or sub-optimally by many research workers in different times. The methods and algorithms mainly include: Integer programming, Dynamic programming, Longest path problem approach, Job sequencing, partitioning and decomposition, Branch and Bound algorithm, Assignment technique etc.

One of the earliest formulations is suggested by Dantzig, Fulkerson and Johnson [3]. The difficulties in finding an optimal tour in solving the integer program are due to the appearance of enormous number of loop constraints. However they overcame the large number of loop constraints by beginning with only a few, and then adding new ones only as they were needed to block sub-tours. Combinatorial arguments were used to eliminate fractional solutions and to find an optimal tour. Finally it was demonstrated that for the problem at hand, an ordinary linear programming could be devised whose solution gave integer valued x_{ij}'s representing the optimal tour. The constraints that rule out some fractional solutions but no integer solutions were forerunners to Gomory's [4] "Cutting plane" constraint for solving any integer linear program. Dynamic programming algorithms have been developed by Bellman [2], Gonzales (1962) and Held and Karp [5].

Several researchers [1, 6, 7 and 8] 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.

II MATHEMATICAL FORMULATION

$$Min \quad Z = \sum_{i=1}^{m} \sum_{j=1}^{m} D(i,j)X(i,j)\ ,\ i,j \in M \qquad \text{..... (3.2.1)}$$

Subject to the constraints:

$$\sum_{i \in \{1\} \cup N_1} \sum_{j \in N_1 \cup H}^{n} X(i,j) = n_1 + 1 \qquad \text{..... (3.2.2)}$$

$$\sum_{i \in \{h\} \cup N_2} \sum_{j \in \{1\} \cup N_2}^{n} X(i,j) = n_2 + 1 \qquad \text{..... (3.2.3)}$$

$$\sum_{i \in M} \sum_{j \in M} X(i,j) = m \qquad \text{..... (3.2.4)}$$

$$X_{ij} = 0 \quad or \quad 1 \qquad \text{..... (3.2.5)}$$

The constraint (3.2.1) represents that the total distance is to be minimum while the travelling salesman touring the m (<n) cities under the consideration.

Constraint (3.2.2) and (3.2.3) indicates that the salesman starts his touring from the home city, visiting n_1-cities before reaching to any one of the sub-headquarter city and then visiting the n_2-cities before reaching to headquarter city. Constraint (3.2.4) indicates that the salesman should visit only m- distinct cities in his tour. Finally the binary variable constraint (3.2.5) X_{ij}=1 represents that the salesman is visited the j^{th} city from i^{th} city, otherwise X_{ij}=0. X also represents a tour for the salesman with m cities. i.e in this tour the salesman visits each of the cities only once.

Here we considered a variant of well-known Travelling Salesman Problem in which a subset with m (<n) cities of n cities has to be traveled by the salesman, i.e. the number of cities to be travelled by a salesman is m. The problem is to find a minimum cost tour by visiting m cities, with given number of cities with the conditions. More specifically, the set of n cities are divided into r sets or clusters such that $N = \{1\} \cup N_1 \cup N_2 \cup H$, here all the subsets are mutually disjoint.

For availing the simplicity in the combinatorial structure of the travelling salesman problem with additional constraints (TSPAC) problem, we developed a Lexi – Search algorithm using Pattern Recognition Technique, which gives an exact optimal solution. The TSPAC problem is illustrated with a suitable numerical example. In particular for combinatorial problems where one has to make the selection of various things, the Lexi – Search approach is more efficient.

A Numerical illustration:

For illustrating the concepts and definitions involved in the problem we have considered N={1,2 , ….,8}, H={4,6}, |N|=n=8, $|N_1|=n_1=2$, $|N_2|=n_2=3$ and |M|=m=7. The cost matrix $C(i, j)$ of TSPAC is given as follows.

Table-1

C(i, j) =		1	2	3	4	5	6	7	8
	1	∞	**10**	**15**	∞	**1**	**20**	**9**	∞
	2	**52**	∞	**0**	**18**	**22**	∞	**19**	**11**
	3	**8**	**31**	∞	**14**	∞	**17**	**49**	**21**
	4	**16**	∞	**4**	∞	**23**	∞	**6**	∞
	5	**2**	**42**	**19**	∞	∞	**26**	∞	**5**
	6	∞	**29**	**34**	∞	**57**	∞	**6**	**43**
	7	**38**	**3**	**33**	**15**	∞	**13**	∞	**10**
	8	∞	**31**	∞	**12**	**16**	**7**	**37**	∞

In the above table the value infinity '∞' indicates the non-connectivity of the cities directly. In this numerical example we are given eight cities and the salesman wants to visiting 7 cities only with the condition that the trip schedule starts from city 1 (headquarter city), visiting 2 before reaching the sub-headquarter city (either 4 or 6) and he will written to home city by visiting 3 cities.

The entire $C(i, j)$'s are taken as non–negative integers but it can be easily seen that this is not a necessary condition and the cost can as well as real quantities. For example $C(3, 6) = 17$, represents the bulk cost of the travelling salesman by visiting the sixth city from third city and $C(5, 4) = \infty$ indicates that the salesman may not visit the city 4 from 5 directly.

An indicator matrix *X = [X (i, j) / X (i, j)= 0 or 1]* in which *X (i, j)=1* indicates that the j^{th} city is visited from i^{th} city, otherwise *X (i, j)=0*. *X* is called a solution. The indicator matrix *X* with 0 or 1 is given in the following table.

Table – 2

X (i, j) =	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**1**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**1**
	0	**0**	**0**	**1**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**1**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**0**	**1**	**0**

The indicator matrix *X* is represented in Table – 2 is feasible. The corresponding ordered pairs of *X* are indicated as ***(1, 3), (2, 1), (3, 5), (4, 8), (5, 4),(7,2),(8, 7)***. The representation of the solution *X (3, 5) =1* to the problem is that the salesman has visited 5^{th} city from 3^{rd} city. For the above example of the feasible allocation set is represented below.

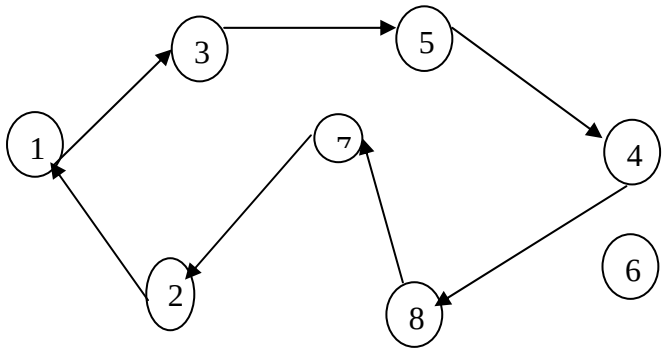

Fig. 2 Feasible Trip schedule of TSPAC

III CONCEPTS AND DEFINITIONS

A Definition of a Pattern:

An indicator two dimensional array *X* which is associated with the selection of items from different clusters is called a "**pattern**". A pattern is said to be feasible if *X* has a feasible solution. The pattern represented in the above Table-2 is a feasible solution.

Now the value of the pattern *X* is defined as follows.

$$V(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} \; X_{ij}$$

The value *V(X)* gives the total cost of the TSPAC of the solution represented by *X*. Thus the value of the feasible pattern gives the total cost. 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 pairs.

B Definition of an Alphabet – Table and a word:

There is **M= n x n** ordered pairs in the two dimensional array *X*. These are arranged in ascending order of their corresponding costs and are indexed from 1 to M (Sundara Murthy – (1979)). Let **SN**= {1, 2… M} be the set of M indices. **D, CD** is the corresponding costs and cumulative sums of the elements of **D**. The arrays **SN, D, CD,**

R, and C are indicate the serial number, cost of the *TSPAC*, cumulative cost, row and column indices respectively. For the numerical example given in Table – 3, if **r ε SN** then [R(r), C(r)] be the cost in their position, D(r)= d[R(r), C(r)] and CD(r)=**ΣD(i),** i= 1, 2, ...…, r.

Table -- 3: ALPPHABET TABLE (AT)

SN	D	CD	R	C
1	**0**	**0**	**2**	**3**
2	**1**	**1**	**1**	**5**
3	**2**	**3**	**5**	**1**
4	**3**	**6**	**7**	**2**
5	**4**	**10**	**4**	**3**
6	**5**	**15**	**5**	**8**
7	**6**	**21**	**4**	**7**
8	**6**	**27**	**6**	**7**
9	**7**	**34**	**8**	**6**
10	**8**	**42**	**3**	**1**
11	**9**	**51**	**1**	**7**
12	**10**	**61**	**1**	**2**
13	**10**	**71**	**7**	**8**
14	**11**	**82**	**2**	**8**
15	**12**	**94**	**8**	**4**
16	**13**	**107**	**7**	**6**
17	**14**	**121**	**3**	**4**
18	**15**	**136**	**1**	**3**
19	**15**	**151**	**7**	**4**
20	**16**	**167**	**4**	**1**

21	**16**	**183**	**8**	**5**
22	**17**	**200**	**3**	**6**
23	**18**	**218**	**2**	**4**
24	**19**	**237**	**2**	**7**
25	**19**	**256**	**5**	**3**
26	**20**	**276**	**1**	**6**
27	**21**	**297**	**3**	**8**
28	**22**	**319**	**2**	**5**
29	**23**	**342**	**4**	**5**
30	**26**	**368**	**5**	**6**
31	**29**	**397**	**6**	**2**
32	**31**	**428**	**3**	**2**
33	**31**	**459**	**8**	**2**
34	**33**	**492**	**7**	**3**
35	**34**	**526**	**6**	**3**
36	**37**	**563**	**8**	**7**
37	**38**	**601**	**7**	**1**
38	**42**	**643**	**5**	**2**
39	**43**	**686**	**6**	**8**
40	**49**	**735**	**3**	**7**
41	**52**	**787**	**2**	**1**
42	**57**	**844**	**6**	**5**
43	∞	∞	**1**	**1**
-	**-**	**-**	**-**	**-**

64	∞	∞	**8**	**8**

Let $\mathbf{L_k} = \{\alpha_1, \alpha_2 \ldots\ldots, \alpha_k\}, \alpha_i \varepsilon \mathbf{SN}$ be an ordered sequence of **k** indices from **SN**. The pattern represented by the ordered pairs whose indices are given by $\mathbf{L_k}$ is independent of the order of α_i in the sequence, the indices are arranged in the increasing order such that $\alpha_i \leq \alpha_{i+1}$, i =1, 2, …., n–1. The ordered sequence $\mathbf{L_k}$ is called a **'sensible word'** if $\alpha_i < \alpha_{i+1}$ for i=1, 2, …., k–1 otherwise we call it as **'Non–sensible word'**. A word $\mathbf{L_k}$ (**k < m**) is said to be partial feasible word if the pattern **X** represented by $\mathbf{L_k}$ is feasible, if **k = m** the word represents a solution. A leader $\mathbf{L_k}$ is said to be feasible if the block of words defined by it contains atleast one feasible word.

Now the value of the word $\mathbf{L_k}$ is defined as follows.

$$\mathbf{V(L_k) = V(L_{k-1}) + D(\alpha_k) \text{ with } V(L_0) = 0}$$

A lower bound LB ($\mathbf{L_k}$) for the values of the block of words represented by $\mathbf{L_k}$ can be defined as follows.

$$\mathbf{LB(L_k) = V(L_k) + CD(\alpha_k + m - k) - CD(\alpha_k)}$$

Consider the partial word $\mathbf{L_4}$ = (1, 4, 8, 12)

$$\mathbf{V(L_4)} = 00 + 03 + 06 + 10 = 19$$

$$\mathbf{LB(L_4)} = V(L_4) + CD(\alpha_4 + 7 - 4) - CD(\alpha_4), \text{ here } \alpha_4 = 12.$$

$$= V(L_4) + CD(15) - CD(12)$$

$$= 19 + 94 - 61 = 52$$

C Feasibility criterion of a partial word:

A recursive algorithm is developed for checking the feasibility of a partial word. $L_{k+1} = (\alpha_1, \alpha_2 \ldots\ldots, \alpha_k, \alpha_{k+1})$ given that L_k is a feasible partial word. We will introduce some more notations which will be useful in the sequel.

IR be an array where IR (i) =1, $i \varepsilon n$ represents that the salesman is visiting the i^{th} city from some other city; otherwise zero.

IC be an array where IC (j) = 1, $j \varepsilon n$ indicates that the salesman is visiting the j^{th} city from some city; otherwise zero.

L be an array where L (i) is the letter in the i^{th} position of a partial word

SW be an array where SW (i)=j represents that the salesman is visiting the j^{th} city from city i, Sw(i)= 0 indicates that the salesman is not visiting any city from city i.

SWI be an array where SWI (i) be the inverse of SW

Then for a given partial word $L_k = (\alpha_1, \alpha_2 \ldots\ldots, \alpha_k)$, the values of the arrays IR, IC, SW, SWI and L are as follows.

IR($R(\alpha_i)$) =1, i=1, 2... k, otherwise IR (j)=0

IC($C(\alpha_i)$) =1, i = 1, 2 …k, otherwise IC (j) =0

SW($R(\alpha_i)$) = $C(\alpha_i)$

SWI($C(\alpha_i)$ = $R(\alpha_i)$

L (i) = α_i, i=1, 2… k, otherwise L(j) =0.

The recursive algorithm for checking the feasibility of a partial word L_k 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 **RA=R (α_k), CA=C (α_k).**

Algorithm 1: (Checking the Feasibility)

Step 0:	IX = 0	goto 1
Step 1:	IS (IR (RA)) = 1)	If Yes goto 10
		If No goto2
Step 2:	IS (IC (CA)) = 1)	If Yes goto 10
		If No goto 2a
Step2a:	IS (ISH=1)	If yes goto 10, If no goto 3
Step 3:	W=IR(RA)	goto 3a
Step3a:	Is W=IC(CA)	If yes goto 9a, If no goto3b
Step 3b:	Is SW (W) =0	If Yes goto 4, Else goto 3c
Step 3c:	W=SW(W)	goto 3a
Step 4:	U=IC(CA)	goto 4a
Step 4a:	Is U=IR(RA)	If Yes goto 9a, Else goto 4b
Step 4b:	Is SWI(U)=0	If Yes goto 5 ,Else goto 4c
Step 4c:	U=SWI(U)	goto 4b
Step 5:	Is W=1	If Yes goto 5a, Else goto 5b
Step 5a:	IDX(SW(W))=1	goto 7
Step5b:	Is U=h	If Yes goto 5c, Else goto 6
Step 5c:	IDX[SWI(U)]=1	goto 7
Step 6:	Is W=h	If Yes goto 6a, Else goto 6b
Step 6a:	IDX(SW(W))=2	goto 8
Step 6b:	Is U=1	If Yes goto 6c, Else goto 6d
Step 6c:	IDX(SWI(U))=2	goto 8
Step 6d:	IDX(W)=3, IDX(SW(W))=3	goto 8b
Step 7:	CNT1=CNT1+1	goto 7a
Step 7a:	Is(CNT1<=n_1)	If Yes goto , Else goto 10
Step 8:	CNT2=CNT2+1	goto 7a
Step 8a:	Is(CNT2<=n_2)	If Yes goto , Else goto 10

Step 8b:	CNT=CNT1+CNT2+1	goto 8c
Step 8c:	Is (CNT<=m)	If Yes goto 9, Else goto 10
Step 9:	IX = 1	
Step9a:	Is k=m	If yes goto 9, Else goto 10
Step 10:	END.	

This recursive algorithm is used in Lexi Search algorithm to check the feasibility of a partial word. We start the algorithm with a large value say '∞' as a trial value 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 the partial word $L_1 = (a_1) = (1)$. A partial word L_k is constructed as $L_k = L_{k-1} * (a_k)$ where * indicates concatenation i.e. chain formation. We will calculate the values of V (L_k) and LB (L_k) simultaneously. Then two cases arise one for branching and the other for continuing the search.

1. **LB (L_k) < VT**. Then we check whether L_k is feasible or not. If it is feasible we proceed to consider a partial word of order (k+1), which represents a sub block of the block of words represented by L_k. If L_k is not feasible then consider the next partial word of order by taking another letter which succeeds a_k in the k^{th} position. If all the words of order 'k' are exhausted then we consider the next partial word of order (k–1).

2. **LB (L_k) ≥ VT.** In this case we reject the partial word L_k. We reject the block of word with L_k as leader as not having optimum feasible solution and also reject all partial words of order 'k' that succeeds L_k.

Now we are in a position to develop a Lexi–Search algorithm to find an optimal feasible word.

IV ALGORITHM -2(LEXI SEARCH ALGORITHM)

Step 0:	(Initialization) The arrays SN, D, CD, R, C and the values N, n1, n2, m, are made available. IR, IC, L, SW, SWI, V and LB are initialized to zero. The values I=1, J=0, VT= ∞, H= N * N – m+k.	
Step 1:	J=J+1	
	IS (J>H)	If Yes goto 12
		If No goto 2
Step 2:	L (I) =J	
	IS (I=1) If Yes V (I) =D (J)	goto 4
		If No goto 3
Step 3:	V (I) =V (I –1) + D (J)	goto 4
Step 4:	LB (I) =V (I) +CD (J+M–I)–CD (J)	goto 5
Step 5:	IS (LB (I) ≥VT)	If Yes goto 12 If No goto 6

Step 6:	RA = R (J)	goto 7
	CA = C (J)	
	SW(W)=0, SWI(U)=0,CNT=0, CNT1=0,CNT2=0	
Step 7:	Check the feasibility of L (using algorithm 1)	
	IS (IX=0)	If Yes goto 8
		If No goto 9
Step 8:	IS (J=H x H)	If Yes goto 13
		If No goto 1
Step 9:	IS (I=M)	If Yes goto 10
		If No goto 11
Step 10:	L (I) =J	
	L (I) is full length word and is feasible	
	VT=V (I), record L (I), VT.	goto 14
Step 11:	IR (RA)=1	
	IC(CA)=1	
	SW(W)=U	
	SWI(U)=W	
	I=I+1	goto 1
Step 12:	IS (I=1)	If Yes goto 16 If No goto 13
Step 13:	I=I –1	goto 14
Step 14:	J=L(I)	
	RA=R(J), CA=C(J)	
	IR(RA)=0, IC(CA)=0	
	SW(W)=0, SWI(U)=0	goto 1

Step 16: stop and end (the current value of VT, when the search terminates it gives the value of an optimal solution).

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

Search Table:

The working details of getting an optimal word using the above algorithm for the illustrative numerical example is given in the following Table-4. The columns named (1), (2), (3), (4), (5), (6) and (7) gives the letters in the first, second, third, fourth, fifth, sixth and seventh places of a word respectively, the corresponding V(I) and LB(I) are indicated in the next two columns. The column R, C gives the row and column indices of the letter. The last column gives the remarks regarding the acceptability/rejectance of the partial words. In the following table A, R indicates Acceptance and Rejectance of a partial word.

Table –4: SEARCH TABLE (ST)

SN	1	2	3	4	5	6	7	V	LB(I)	R	C	REMARKS
1	01							0	21	2	3	A
2		02						1	21	1	5	A
3			03					3	21	5	1	R(semi cycle)
4			04					4	25	7	2	A
5				05				8	25	4	3*	R
6				06				9	28	5	8	A
7					07			15	28	4	7	A
8						08		21	28	6	7*	R
9						09		22	30	8	6*	R
10						10		23	32	3	1	A
11							11	32	32	1*	7	R
12							12	33	33	1*	2	R
13							13	33	33	7*	8	R
14							14	34	34	2*	8	R
15							15	35	35=VT	8	4	A(complete)
16						11		24	34	1*	7	R
17						12		25	35=VT	1*	2	R
18					08			15	30	6	7	A
19						09		22	30	8	6	A
20							10	30	30=VT*	3	1	A(Complete)
21						10		23	32>VT			R
22					09			16	33>VT			R
23				07				10	31>VT			R
24			05					05	29	4	3*	R
25			06					06	33>VT			R
26		03						02	26	5	1	A
27			04					05	26	7	2	R(>m)
28			05					06	30=VT			R
29		04						03	31>VT			R
30	02							01	27	1	5	A

31		**03**						**03**	**27**	**5**	**1**	**R(SC)**
32		**04**						**04**	**32>VT**			**R**
33	**03**							**02**	**33>VT**			**R**

At the end of the search the current value of VT is 30 and it is the value of the optimal feasible word L_7 = (1, 2, 4, 6, 8, 9, 10). It is given in the 20th row of the search table. The array IR, IC, SW, SWI, L and JN 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**	**7**	**8**
IR	**1**	**1**	**1**	**-**	**1**	**1**	**1**	**1**
IC	**1**	**1**	**1**	**-**	**1**	**1**	**1**	**1**
SW	**5**	**3**	**1**	**-**	**8**	**7**	**2**	**6**
SWI	**3**	**7**	**2**	**-**	**1**	**8**	**6**	**5**
L	**1**	**2**	**4**	**6**	**8**	**9**	**10**	
IDX	**-**	**2**	**2**	**-**	**1**	**-**	**2**	**1**

TABLE-6

X (i, j) =	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
	0	**0**	**1**	**0**	**0**	**0**	**0**	**0**
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**1**
	0	**0**	**0**	**0**	**0**	**0**	**1**	**0**
	0	**1**	**0**	**0**	**0**	**0**	**0**	**0**
	0	**0**	**0**	**0**	**0**	**1**	**0**	**0**

The tour represented by the above pattern is [(2, 3), (1, 5), (7, 2), (5, 8), (6, 7), (8, 6), (3, 1)], where the salesman starts the tour from the headquarter city {1}, visits the n_1 -{5, 8}-cities before reaching one of the sub-headquarter city {6} and then returns the home city by visiting the n_2-{7, 2, 3}-cities. In his tour the optimal solution for the considerable numerical example is 30. It also can be represented by $1 \rightarrow 5 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow 3 \rightarrow 1$. The diagrammatic representation of this arrangement can also see in the following figure.

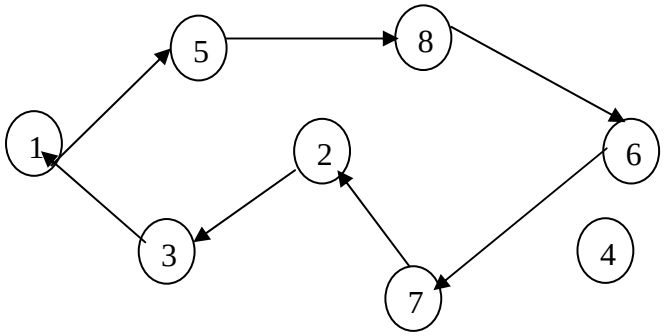

Fig. 2 Optimal Trip schedule of TSPAC

Experimental Results:

A Computer program for the proposed Lexi – Search Algorithm is written in C language and is tested. The experiments are carried out on a COMPAQ (dx2280 MT) system by generating the distance values (D_{ij}) uniformly between [1, 1000]. We tried a set of problems for different sizes. Random numbers are used to construct the Time matrix. The results are tabulated in Table-7. For each instance, five data sets are tested. It is seen that the time required for the search of the optimal solution is fairly less. The following table shows that the CPU runs time taken by the proposed LSA to find the optimal solution of various hard instances.

Table: 7

SN	N	N_1	N_2	SH	NPT	CPU Run Time in seconds with the proposed LSA		Total time Avg. (AT+ ST)
						Avg. AT	Avg. ST	
1	6	2	2	1	6	0.0000	0.0000	0.0000
2	8	2	3	2	6	0.0000	0.0000	0.0000
3	10	3	4	2	6	0.0549	0.0000	0.0549
4	12	4	5	2	6	0.1098	0.0000	0.1098
5	15	5	6	3	6	0.2197	0.2747	0.4944
6	18	10	5	2	6	0.2747	0.2197	0.4944
7	20	6	10	3	6	0.3846	0.1648	0.5494
8	22	8	10	3	6	0.4945	0.3296	0.8241
9	25	12	10	2	6	0.6043	3.6264	4.2307
10	30	10	10	9	6	0.8241	5.1648	5.9889

** The time is represented in seconds. In the table-7 SN = serial number, N = number of cities, N_1 = number of cities to be visited before sub headquarter, N_2= number of cities to be visited after sub headquarter, SH= number of cities to in sub headquarter set, NPT= number of problems tried, AT = CPU run time for printing the alphabet table, ST=CPU run time for obtaining the optimal solution of the search table.

Conclusion:

In this chapter, we have developed a Lexi-search algorithm based on pattern recognition technique to solve the TSPAC. The model is then formulated as a zero-one programming problem. A Lexi-Search Algorithm based on Pattern Recognition Technique is developed for getting an optimal solution. A suitable numerical example is quoted for better understood the concepts and the steps involved in the algorithm. We programmed the proposed algorithm using C-language. The computational details are reported. As an observation the CPU run time is fairly less for higher dimensional problem, it gives an optimal solution. Moreover, Lexi-search algorithms are proved to be more efficient in many combinatorial problems. Many researchers have used different types of alphabet tables and have shown that their Lexi-search algorithms are efficient and are faster. Srinivas -1989 proved that lexi-search with pre-processing is very effective in solving ASP (assignment problem) and Traveling salesman problem. Based on this experience we strongly feel that this algorithm can perform larger size problems and more over it is very efficient.

V ACKNOWLEDGEMENTS

The first 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.

VI REFERENCES

[1] Bhavani, V. and Sundara Murthy, M.(2005):Time-Dependent Traveling Salesman Problem OPSEARCH 42, PP. 199-227.
[2] Bellman, R and Dreyfus, S (1962) : Applied Dynamic programming - Princeton University press, Princeton, New jersey
[3] Dantzig GB. (1951) : Application of the simplex method to a transportation problem. Activity analysis of production and allocation. Cowles Commission Monograph 13.1951.
[4] Gomory, R.E (1963) : "An algorithm for Integer Solution to Linear Programs"Recent Advances in Mathematics rogramming269-302Mc.GrawHill,New York
[5] Held, M. and R.H. Karp (1962) : The TSP and Minimal Spanning Tree" Opns.Res., Vol. 18, No. 6, p 1138
[6] Pandit, S.N.N. and Srinivas, K(1962): A Lexisearch algorithm for traveling Salesman problem, IEEE, 2521-2527.
[7] Pandit, S.N.N . and Rajbhougshi(1976):Restricted TSP through n sets of nodes. Paper presented at the 9th Annual Convention of ORSI, Calcutta
[8] Sundara Murthy, M(1979):Combinatorial Programming - A Pattern Recognition Approach. PhD, Thesis REC, Warangal, India