A NEW APPROACH FOR VARIANT MULTI ASSIGNMENT PROBLEM

[1]SOBHAN BABU.K [2]CHANDRA KALA.K [3]PURUSHOTTAM.S [4]SUNDARA MURTHY.M

[1]Assistant Professor, Department of Mathematics, UCE, JNTUK, Kakinada, A.P., India.

[2]Assistant System Engineer, Tata Consultancy Services, Hyderabad, A.P., India.

[3]Research Scholar, Department of Mathematics, Sri Venkateswara University, Tirupati, A.P., India.

[4]Professor, Department of Mathematics, Sri Venkateswara University, Tirupati, A.P., India.

Abstract— A large number of real-world planning problems called Combinatorial Optimization 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- Assignment Problem, Lexi-Search, Pattern Recognition, Alphabet Table, Search Table.

I. INTRODUCTION

In this paper we study a problem called "Three Dimensional Variant Multi Assignment Problem". (TDVMAP). Let N = (1,2, . . .,n) be the set of n persons/agents, J = (1,2, . . .,m) be the set of m jobs/tasks and K= (1,2, . . ., k) be the set of k facilities. In this problem we assign the set of jobs to the set of persons under some restrictions. The subset of $N_i \subseteq N$ persons will be assigned l_i jobs each where $\cup N_i = N$, $|N_I| = n_i$, i = 1,2,…n. The objective is to find the total minimum cost of assigning the jobs to the persons, with the restriction that each job should be assigned to only one person and if a person is assigned more than one job then they should be at the same facility.

II. MATHEMATICAL FORMULATION

$$\text{Minimize } Z = \sum_{i\in N}\sum_{j\in J}\sum_{k\in K} C(i,j,k)\, X(i,j,k) \quad \text{-------- (1)}$$

$$\sum_{i\in N_i}\sum_{j\in J}\sum_{k\in K} \mathbf{X(i,j,k)} = \mathbf{n_i . l_i} \quad \text{i=1,2..p} \quad \text{--------- (2)}$$

$$\sum_{i\in N}\sum_{k\in K} \mathbf{X(i,j,k)} = \mathbf{1} \qquad \forall\, \mathbf{j} \in \mathbf{J} \quad \text{---------- (3)}$$

$$X(i,j,k) = o \text{ or } 1 \quad \text{---------- (4)}$$

If $X(i_1,j_1,k_1) = X(i_2,j_2,k_2) = 1$, $i_1 = i_2$ & $j_1 \neq j_2$ then $k_1 = k_2$. ---------- (5)

Here **| N | = n,** $|N_i| = n_i$, $\sum_{i=1}^{p} n_i = n$, **| J | = m,** $\sum_{i=1}^{p} n_i l_i = m$

Constraint (2) indicates that l_i jobs are assigned to the each of n_i persons and (3) represents each job is assigned to only one person. The restriction (5) indicates that if a person is assigned to different jobs it should be at the same facility. The problem is to find the total minimum cost of assigning the jobs to the persons with the required restrictions.

In the sequel we developed a Lexi-search algorithm based on the "Pattern Recognition Technique" to solve this problem which takes care of simple combinatorial structure of the problem and computational results are reported.

III. NUMERICAL ILLUSTRATION

The concepts and the algorithm developed will be illustrated by a numerical example for which n=6(persons), m=10 (jobs), and k = 2(facilities) then the matrix is given as follows. In this problem we have to assign any one person (agent) can do any three jobs (tasks), any two persons (agents) can do any two jobs (tasks) and any three persons (agents) can do any one job (task).

TABLE-I

$$C(i,j,1) = \begin{bmatrix} 10 & 2 & 14 & 9 & 6 & 7 & 21 & 32 & 18 & 11 \\ 7 & 12 & 9 & 3 & 5 & 6 & 9 & 16 & 54 & 12 \\ 4 & 8 & 6 & 12 & 21 & 9 & 21 & 14 & 45 & 13 \\ 21 & 9 & 12 & 9 & 32 & 10 & 19 & 25 & 16 & 10 \\ 10 & 12 & 30 & 15 & 12 & 17 & 30 & 12 & 12 & 9 \\ 15 & 7 & 34 & 17 & 7 & 16 & 14 & 17 & 9 & 5 \end{bmatrix}$$

$$C(i,j,2) = \begin{bmatrix} 21 & 11 & 16 & 9 & 15 & 10 & 12 & 32 & 26 & 16 \\ 14 & 15 & 20 & 10 & 16 & 3 & 6 & 9 & 21 & 14 \\ 9 & 17 & 11 & 31 & 21 & 16 & 7 & 9 & 10 & 11 \\ 16 & 23 & 8 & 15 & 10 & 3 & 1 & 3 & 20 & 23 \\ 12 & 40 & 14 & 36 & 9 & 21 & 14 & 19 & 4 & 13 \\ 8 & 18 & 9 & 42 & 8 & 11 & 19 & 9 & 32 & 20 \end{bmatrix}$$

IV. CONCEPTS & DEFINITIONS

A. Definition of a Pattern

An indicator three-dimensional array which is associated with an assignment is called a 'pattern'. A Pattern is said to be feasible if X is a solution. The pattern represented in the table-2 is a feasible pattern. Now T(X) the value of the pattern X is defined as

$$T(X) = \max_{i \in I} \sum_{j \in J} \sum_{k \in K} T(i,j,k)\, X(i,j,k)$$

The value T(X) gives the total time of the assignment for the solution represented by X. Thus the value of the feasible pattern gives the total time 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,k)] for which X(i,j,k)=1, with understanding that the other X(i,j,k)'s are zeros.

The ordered triple set [(4,7,2), (1,2,1), (2,4,1), (4,6,2), (4,8,2), (3,1,1), (5,9,2), (2,5,1),(6,10,1),(3,3,1))] represents the pattern given in the table-2, which is a feasible solution.

TABLE-II

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

$$X(i,j,2) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

There is M = m×n×p ordered triples in the three dimensional array X. For convenience these are arranged in ascending order of their corresponding times are adding indexed from 1 to M (Sundara Murthy M-1979). Let SN = (1, 2. . . M) be the set of M indices. Let TD be the corresponding array of times. If a, b ∈ SN and a < b the TD (a) ≤ TD (b). Also let the arrays R, C, F be the array or row, column and facility indices of the ordered triples represented by SN and CT be the array of cumulative sum of the elements of TD. The arrays SN, TD, CT, R, C, F for the numerical example are given in the table-3. If p ∈ SN then (R (p), C (p), F (p)) is the ordered triple and TD (a) = T(R (a), C (a), F (a)) is the value of the ordered triple and CT (a) =$\sum_{i=1}^{a} TD(i)$.

TABLE-III **(ALPHABET TABLE)**

S.No	TD	CT	R	C	F
1	1	1	4	7	2
2	2	3	1	2	1
3	3	6	2	4	1
4	3	9	2	6	2
5	3	12	4	6	2
6	3	15	4	8	2
7	4	19	3	1	1
8	4	23	5	9	2
9	5	28	2	5	1
10	5	33	6	10	1
11	6	39	1	5	1
12	6	45	2	6	1
13	6	51	3	3	1
14	6	57	2	7	2
15	7	64	1	6	1
16	7	71	2	1	1
17	7	78	6	2	1
18	7	85	6	5	1
19	7	92	3	7	2
20	8	100	3	2	1
21	8	108	4	3	2
22	8	116	6	1	2
23	8	124	6	5	2
24	9	133	1	4	1
25	9	142	2	3	1
26	9	151	2	7	1
27	9	160	3	6	1
28	9	169	4	2	1
29	9	178	4	4	1
30	9	187	5	10	1
31	9	196	6	9	1
32	9	205	1	4	2
33	9	214	2	8	2
34	9	223	3	1	2
35	9	231	3	8	2
36	9	240	5	5	2
37	9	249	6	3	2
38	9	258	6	8	2
39	10	268	1	1	1
40	10	278	4	6	1
41	10	288	4	10	1
42	10	298	5	1	1
43	10	308	1	6	2
44	10	318	2	4	2
45	10	328	3	9	2
46	10	338	4	5	2
47	11	349	1	10	1
48	11	360	1	2	2
49	11	371	3	3	2
50	11	382	3	10	2
51	11	393	6	6	2
52	12	405	2	2	1
53	12	417	2	10	1
54	12	429	3	4	1

55	12	441	4	3	1
56	12	453	5	2	1
57	12	465	5	5	1
58	12	477	5	8	1
59	12	489	5	9	1
60	12	501	1	7	2
61	12	513	5	1	2
62	13	526	3	10	1
63	13	539	5	10	2
64	14	553	1	3	1
65	14	567	3	8	1
66	14	581	6	7	1
67	14	595	2	1	2
68	14	609	2	10	2
69	14	623	5	3	2
70	14	637	5	7	2
71	15	652	5	4	1
72	15	667	6	1	1
73	15	682	1	5	2
74	15	697	2	2	2
75	15	712	4	4	2
76	16	728	2	8	1
77	16	744	4	9	1
78	16	760	6	6	1
79	16	776	1	3	2
80	16	792	1	10	2
81	16	808	2	5	2
82	16	824	3	6	2
83	16	840	4	1	2
84	17	857	5	6	1
85	17	874	6	4	1
86	17	891	6	8	1
87	17	908	3	2	2
88	18	926	1	9	1
89	18	944	6	2	2
90	19	963	4	7	1
91	19	982	5	8	2
92	19	1001	6	7	2
93	20	1021	2	3	2
94	20	1041	4	9	2
95	20	1061	6	10	2
96	21	1082	1	7	1
97	21	1103	3	5	1
98	21	1124	3	7	1
99	21	1145	4	1	1
100	21	1166	1	1	2
101	21	1187	2	9	2
102	21	1208	3	5	2
103	21	1229	5	6	2
104	23	1252	4	2	2
105	23	1275	4	10	2
106	25	1300	4	8	1
107	26	1326	1	9	2
108	30	1356	5	3	1
109	30	1386	5	7	1
110	31	1417	3	4	2
111	32	1449	1	8	1
112	32	1481	4	5	1
113	32	1513	1	8	2
114	32	1545	6	9	2
115	34	1579	6	3	1
116	36	1615	5	4	2
117	40	1655	5	2	2
118	42	1697	6	4	2
119	45	1742	3	9	1
120	54	1796	2	9	1

Let us consider 21 ∈ SN. It represents that the ordered triple (R (21), C (21), F (21)) = (4, 3, 2). Then TD (21) = T (4, 3, 2) = 8 and CT (21) = 108.

B.Definition of Alphabet Table and Word

Let SN = (1,2,…) be the set of indices, TD be an array of corresponding costs of the ordered triples and CT be the array of cumulative sums of elements in TD. Let arrays R, C and F be respectively, the row, column and facility indices of the ordered triples. Let $L_k = \{a_1, a_2, - - -- - , a_k\}$, $a_i \in SN$ be an ordered sequence of k indices from SN. The pattern represented by the ordered triples whose indices are given by L_k is independent of the order of a_i in the sequence. Hence for uniqueness the indices are arranged in the increasing order such that $a_i \leq a_{i+1}$, i = 1, 2, - - - -, k-1. The set SN is defined as the "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 called a "sensible word". If $a_i < a_{i+1}$, for i =1,2, - - - -, 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 pattern. 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. Our interest is only in set of words of length atmost equation, 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 said to be feasible, if the block of word, defined by it has at least one feasible word.

C.Value of the Word

The value of the (partial) word L_k, V (L_k) is defined recursively as V (L_k) = V (L_{k-1}) + TD (a_k) with V (L_o) = 0 where TD (a_k) is the cost array arranged such that TD (a_k) < TD (a_{k+1}). V (L_k) and V(x) the values of the pattern X will be the same. Since X is the (partial) pattern represented by L_k, (Sundara Murthy – 1979).

D.Search-Table

The working details of getting an optimal word using the above algorithm for the illustrative numerical example is given in the Table-4. The columns named (1), (2), (3),…, gives the letters in the first, second, third and so on places respectively. The columns R, C and F give the row, column and facility 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-IV **(SEARCH-TABLE)**

SN	1	2	3	4	5	6	7	8	9	10	R	C	F	REM
1	1										4	7	2	A
2		2									1	2	1	A
3			3								2	4	1	A
4				4							2	6	2	R
5				5							4	6	2	A
6					6						4	8	2	A
7						7					3	1	1	A
8							8				5	9	2	A
9								9			2	5	1	A
10									10		6	10	1	A
11										11	1	5	1	R
12										12	2	6	1	R
13										13	3	3	1	A=VT(36)
14									11		1	5	1	R > VT
15								10			6	10	1	R > VT
16							9				2	5	1	R > VT
17						8					5	9	2	R > VT
18					7						3	1	1	R > VT
19				6							4	8	2	R > VT
20			4								2	6	2	R = VT
21		3									2	4	1	R > VT
22	2										1	2	1	R > VT

At the end of search table the trail value is 36. The partial word is L_{10} = (1, 2, 3, 5, 6, 7, 8, 9, 10, 13) is a feasible partial word. For this partial word the array IR, IC, IT, LW are given in the following Table – 5.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
LN	1	2	2	3	1	1	-	-	-	-
K	1	1	1	2	2	1	-	-	-	-

In this numerical example first, fifth & sixth persons/agents done only one different job/task, second & third persons/agents done two different jobs/tasks, fourth person/agent done three different jobs/tasks.

TABLE-V

	1	2	3	4	5	6	7	8	9	10
IR	4	1	2	4	4	3	5	2	6	3
IC	7	2	4	6	8	1	9	5	10	3
IF	2	1	1	2	2	1	2	1	1	1
LW	1	2	3	5	6	7	8	9	10	13

At the end of the search the current value of VT is 36 and it is the value of optimal feasible word. L_{10} = (1, 2, 3, 5, 6, 7, 8, 9, 10, 13). At the end of the search table the solution is 36 and the assignment schedule represented by 7^{th} job done by 4^{th} person by using facility 2, 2^{nd} job done by 1^{st} person by using facility 1, 4^{th} job done by 2^{nd} person by using facility 1, 6^{th} job done by 4^{th} person by using facility 2, 8^{th} job done by 4^{th} person by using facility 2, 1^{st} job done by 3^{rd} person by using facility 1, 9^{th} job done by 5^{th} person by using facility 2, 5^{th} job done by 2^{nd} person by using facility 1, 10^{th} job done by 6^{th} person by using facility 1, 3^{rd} job done by 3^{rd} person by using facility 1.

V. COMPUTATIONAL EXPERIENCE

Ross & Soland have developed the branch and bound algorithm conventional to that of Dakin, calculating bound in part by solving binary knapsack problems. They observed that the bound so calculated is identical to the one provided by the Lagrangian relaxation for the two dimensional assignment problem. While Fisher & Jaikumar have provided the bounds dominating the Ross & Soland bounds, Mortello & Toth have proved that their algorithm MTG is faster than both the above. Ravikumar has formulated Lexi-Search Data Guided Algorithm and showed that his algorithm is faster than MTG in many cases. Ramana.V.V.V & Umashankar have considered the GAP as it was considered by Ross & Soland and developed the Lexi-Search algorithm using Pattern Recognition Approach. Except Ramana .V.V.V & Umashankar and Ross & Soland algorithms, all the others considered the maximization version of the problem, with the direct reference to knapsack problems, using profit and weights. The constraint that all agents are to be assigned is relaxed by all of them including Ross & Soland. Ramana & Umashankar have tested the Lexi-Search algorithm considering this constraint and also relaxing it. Ramana & Umashankar have tested the algorithm MTG of Martello & Toth for the same set of problems, with and without backtracking. Ramana & Umashankar have formulated the Lexi-Search Algorithm and showed that their algorithm is faster than the above cases. Around 60 randomly formulated problems of varying sizes are tested with these algorithms on a core 2 duo. We have tested our algorithm with Ramana & Umashankar for the same set of problems, with and with out backtracking. The results are tabulated in tables 6 – 14

And the results are tabulated in Table. Table 6 gives the CPU times for Martello & Toth's MTG with and without backtracking. Table 8 gives the CPU time taken by Ramana & Umashankar Lexi-Search algorithm for two dimensional problems with the constraint on agents in the GAP problem, while table 7 gives the CPU times taken when the constraint is relaxed (equivalent to MTG). Values in the brackets against in each problem set indicate the times taken for solving the problem. After excluding the time taken for sorting the cost array. From table 6 and 7, it can be clearly seen that the time taken by Lexi-Search algorithm is less as compared to the taken by MTG, in most of the cases. Further, from table 7, it is clear that the time taken by Lexi-Search algorithm falls down significantly, when the sort time is excluded. Further 12 & 14 clearly show that the change in alphabet arrangement (primarily the sort procedure adopted by LEXI2) brings down the times

further. When the restriction (that all agents are to be assigned) is considered, the time taken by the problem increased slightly for problems of smaller sizes and when the numbers of tasks are more, the algorithm takes more time. For each type, three data sets are tested. It is seen that time required for the search of execution time of the optimal solution is fairly less.

TABLE-VI

TIME TAKEN BY THE ALGORITHM MTG OF MARTELLO & TOTH

No. of			Total time taken (in sec)					
Agents	*Tasks*	*Problem*	*No-backtracking*			*backtracking*		
			Min.	*Max.*	*Avg.*	*Min.*	*Max.*	*Avg.*
4	*6*	*6*	*.001*	*.001*	*.0010*	*.001*	*.005*	*.0030*
5	*5*	*6*	*.001*	*.006*	*.0035*	*.004*	*.10*	*.0070*
5	*10*	*6*	*.007*	*.013*	*.0100*	*.007*	*.014*	*.0105*
5	*20*	*6*	*.014*	*.026*	*.0200*	*.014*	*.026*	*.0200*
10	*10*	*6*	*.014*	*.014*	*.0140*	*.014*	*.015*	*.0145*

TABLE-VII

TIME TAKEN BY THE LEX1 ALGORITHM OF RAMANA & UMASANKAR WITH OUT CONSTRAINTS ON AGENTS

o.of			*Total Time Taken (in Sec)*					
AGENTS	*TASKS*	*PROBLEMS*	*1st Solution*			*Total Time Taken*		
			Min	*Mac*	*Avg,*	*Min*	*Max*	*Avg.*
4	*6*	*12*	*.001 (.000)*	*.002 (.001)*	*.0015 (.005)*	*.001 (.000)*	*.002 (.001)*	*.0015 (.0005)*
5	*5*	*12*	*.001 (.000)*	*.002 (.001)*	*.0015 (.0005)*	*.001 (.001)*	*.002 (.001)*	*.0015 (.0010)*
5	*10*	*6*	*.003 (.000)*	*.005 (.001)*	*.004 (.0005)*	*.003 (.000)*	*.005 (.001)*	*.0040 (.0005)*
5	*20*	*6*	*.014 (.000)*	*.018 (.001)*	*.0160 (.0005)*	*.014 (.000)*	*.020 (.004)*	*.0170 (.0025)*
10	*10*	*6*	*.015 (.000)*	*.18 (.001)*	*0.0165 (0.000)*	*.015 (.000)*	*.019 (.001)*	*.0170 (.0005)*

TABLE-VIII

TIME TAKEN BY THE LEX1 ALGORITHM OF RAMANA & UMASANKAR WITH CONSTRAINTS ON AGENTS

No.of			*Total Time Taken (in Sec)*					
AGENTS	*TASKS*	*PROBLEMS*	*1st Solution*			*Total Time Taken*		
			Min	*Mac*	*Avg,*	*Min*	*Max*	*Avg.*
4	*6*	*12*	*.001 (.000)*	*.003 (.002)*	*.0020 (.001)*	*.001 (.001)*	*.004 (.003)*	*.0025 (.0015)*
5	*5*	*12*	*.001 (.000)*	*.004 (.003)*	*.0025 (.0015)*	*.001 (.001)*	*.006 (.005)*	*.0035 (.0030)*
5	*10*	*6*	*.004 (.000)*	*.030 (.025)*	*.0170 (.0125)*	*.004 (.000)*	*.555 (.550)*	*.2795 (.2250)*
5	*20*	*6*	*.015 (.000)*	*.023 (.007)*	*.0190 (.0035)*	*.015 (.000)*	*20.81 (20.78)*	*10.41 (10.39)*
10	*10*	*6*	*----*	*(-----*	*----*	*.129 (.111)*	*.174 (.156)*	*.0150* (.133)**

(* NO FEASIBLE SOLUTION FOUND)

TABLE-IX

TIME TAKEN BY THE LEX2 ALGORITHM OF RAMANA & UMASANKAR WITH OUT CONSTRAINTS ON AGENTS

No.of			*Total Time Taken (in Sec)*					
AGENTS	*TASKS*	*PROBLEMS*	*1st Solution*			*Total Time Taken*		
			Min	*Mac*	*Avg,*	*Min*	*Max*	*Avg.*
4	*6*	*12*	*.000 (.000)*	*.001 (.000)*	*.0005 (.000)*	*.000 (.000)*	*.001 (.001)*	*.0005 (.0000)*
5	*5*	*12*	*.001 (.000)*	*.002 (.000)*	*.0010 (.0000)*	*.001 (.001)*	*.001 (.000)*	*.0010 (.0000)*
5	*10*	*6*	*.003 (.000)*	*.004 (.000)*	*.0035 (.0000)*	*.003 (.000)*	*.004 (0.000)*	*.0035 (.0000)*
5	*20*	*6*	*.011 (.000)*	*.015 (.001)*	*.0130 (.0005)*	*.012 (.000)*	*.018 (.004)*	*.0150 (.0020)*
10	*10*	*6*	*.011 (.000)*	*.015 (.000)*	*.0135 (.000)*	*.012 (.000)*	*.015 (.000)*	*.0135 (.000)*

TABLE-X

TIME TAKEN BY THE LEX2 ALGORITHM OF RAMANA & UMASANKAR WITH CONSTRAINTS ON AGENTS

No.of			*Total Time Taken (in Sec)*					
AGENTS	*TASKS*	*PROBLEMS*	*1st Solution*			*Total Time Taken*		
			Min	*Mac*	*Avg,*	*Min*	*Max*	*Avg.*
4	*6*	*12*	*.000 (.000)*	*.002 (.000)*	*.0010 (.0000)*	*.000 (.000)*	*.002 (.000)*	*.0010 (.0000)*
5	*5*	*12*	*.001 (.000)*	*.003 (.000)*	*.0020 (.0000)*	*.001 (.001)*	*.006 (.004)*	*.0030 (.0025)*
5	*10*	*6*	*.003 (.000)*	*.029 (.025)*	*.0160 (.0125)*	*.004 (.000)*	*.356 (.350)*	*.1800 (.1700)*
5	*20*	*6*	*.012 (.000)*	*.020 (.007)*	*.0170 (.0035)*	*.014 (.000)*	*20.72 (20.08)*	*10.367 (10.33)*
10	*10*	*6*	*----*	*----*	*-----*	*.125 (.108)*	*.163 (.147)*	*.0144* (.127)**

(* NO FEASIBLE SOLUTION FOUND)

TABLE-XI

TIME TAKEN BY THE LEX1 OUR PROPOSED ALGORITHM WITH OUT CONSTRAINTS ON AGENTS

o.of AGENTS	o.of TASKS	PROBLEMS	Total Time Taken (in Sec)					
			1st Solution			Total Time Taken		
			Min	Max	Avg.	Min	Max	Avg.
8	12	12	.000 (.000)	.002 (.001)	.001 (.005)	.001 (.000)	.002 (.001)	.0015 (.0005)
10	10	12	.000 (.000)	.002 (.001)	.001 (.0005)	.001 (.001)	.002 (.001)	.0015 (.0010)
10	20	6	.000 (.000)	.005 (.001)	.0025 (.0005)	.005 (.002)	.007 (.004)	.006 (.003)
10	30	6	.008 (.000)	.0010 (.001)	.009 (.0005)	.012 (.001)	.016 (.004)	.014 (.0025)
20	20	6	.010 (.000)	.012 (.001)	0.011 (0.000)	.015 (.000)	.019 (.001)	.0170 (.0005)

TABLE-XII

TIME TAKEN BY THE LEX1 OUR PROPOSED ALGORITHM WITH CONSTRAINTS ON AGENTS

No.of AGENTS	No.of TASKS	PROBLEMS	Total Time Taken (in Sec)					
			1st Solution			Total Time Taken		
			Min	Mac	Avg,	Min	Max	Avg.
8	12	12	.003 (.001)	.006 (.003)	.0045 (.002)	.003 (.002)	.008 (.004)	.0055 (.003)
10	10	12	.001 (.000)	.004 (.003)	.0025 (.0015)	.002 (.003)	.009 (.005)	.0055 (.004)
10	20	6	.004 (.000)	.030 (.025)	.0170 (.0125)	.004 (.000)	.796 (.550)	.400 (.2250)
10	30	6	.018 (.002)	.024 (.007)	.021 (.0045)	.015 (.000)	20.81 (20.78)	10.41 (10.39)

TABLE-XIII

TIME TAKEN BY THE LEX2 OUR PROPOSED ALGORITHM WITH OUT CONSTRAINTS ON AGENTS

No.of AGENTS	No.of TASKS	PROBLEMS	Total Time Taken (in Sec)					
			1st Solution			Total Time Taken		
			Min	Mac	Avg,	Min	Max	Avg.
8	12	12	.001 (.000)	.001 (.000)	.001 (.000)	.001 (.000)	.002 (.001)	.0015 (.0000)
10	10	12	.001 (.000)	.002 (.000)	.0010 (.0000)	.001 (.001)	.001 (.000)	.0010 (.0000)
10	20	6	.006 (.000)	.007 (.000)	.0065 (.0000)	.005 (.000)	.008 (0.000)	.0065 (.0000)
10	30	6	.014 (.000)	.019 (.001)	.0165 (.0005)	.018 (.000)	.020 (.004)	.019 (.0020)
20	20	6	.014 (.000)	.016 (.000)	.015 (.000)	.012 (.000)	.015 (.000)	.0135 (.000)

TABLE-XIV

TIME TAKEN BY THE LEX2 OUR PROPOSED ALGORITHM WITH CONSTRAINTS ON AGENTS

No.of AGENTS	No.of TASKS	PROBLEMS	Total Time Taken (in Sec)					
			1st Solution			Total Time Taken		
			Min	Max	Avg.	Min	Max	Avg.
8	12	12	.002 (.000)	.004 (.000)	.003 (.0000)	.002 (.000)	.004 (.000)	.003 (.0000)
10	10	12	.002 (.000)	.005 (.000)	.0035 (.0000)	.004 (.001)	.006 (.004)	.0050 (.0025)
10	20	6	.012 (.000)	.018 (.025)	.015 (.0125)	.004 (.000)	.356 (.350)	.1800 (.1700)
10	30	6	.012 (.000)	.020 (.007)	.0170 (.0035)	.014 (.000)	20.72 (20.08)	10.367 (10.33)

VI. CONCLUSION

The problems are solved by using the Lexi-Search algorithm based on the Pattern Recognition Technique, with and without the restriction on agents considered in the GAP and studied further by excluding the sort times taken by the respective problems (with backtracking in all the cases). The same problems have been tested with the MTG algorithm of Martello & Toth as well as Ramana & Umashankar, with and without backtracking. Our algorithm is faster than the MTG and Ramana & Umashankar Lexi-Search algorithm in most of the cases. Even with the restriction imposed, the Lexi-Search algorithm takes reasonably less time. Further it is observed that with the modification of the sort procedure while arranging the alphabet table in LEXI 1, the Lexi-Search algorithm LEXI 2 is becoming more efficient. On the whole, it is felt that Our Lexi-Search algorithm is faster than the MTG algorithm as well as Ramana & Umashankar's Lexi-Search algorithm.

ACKNOWLEDGEMENTS

The authors are very much thankful to the referees for their suggestions & useful comments.

REFERENCES

[1] Balachandran, V."An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Network," Operations Research, 24, 1976, pp. 742-759.

[2] Cattrysee, D.G. & Wassenhove, L.N.V. "A Survey of Algorithms for the Generalized Assignment Problems," Euro.J.O.R. 60, 1992, pp.260-272.

[3] Dakin, R. "A Tree Search Algorithm for Mixed Integer Programming Problems," Computer Journal 8, 1965, pp. 250-255.

[4] Devaio & Roved. "An All Zero-One Algorithm for A class of Transportation Problem," Operations Research, 19, 1971, pp.1406-1418.

[5] Fisher, M.L., Jaikumar, R. & Luk N.Van.Wassenhove., "A Multiplier Adjustment Method for the Generalized Assignment Problem," Management Science, 32, 1986, pp.1095-1103.

[6] Geoffrion, A.M., & Graves, G.W. "Multi Commodity Distribution System Design by Benders Decomposition," Management Science, 20, 1974, pp.822-844.

[7] Kuhn, H. "Variants of the Hungarian Method for Assignment Problems," Naval Research Logistics, Qtrly., 3, 1956, pp.253-258.

[8] Martello, S. & Toth, P. "An algorithm for the Generalized Assignment Problem," in J.P.BRANS(ed.), Operations Research, 81,1981, North Holland, pp. 589-603.

[9] Ross, G.T. & Soland, R.M. "A Branch and Bound Algorithm for Generalized Assignment Problem," Mathematical Programming, 1975, pp.91-103.

[10] Ross, G.T. & Soland, R.M. "Modelling Facility Location Problems as Generalized Assignment Problems," Management Science, 24, 1977, pp.345-357.

[11] Ravi Kumar, M. "Data Guided Algorithm in Combinatorial Optimization," 1994, A Ph.D. Thesis, Osmania University, Hyderabad.

[12] 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.

[13] Srinivasan, V. & Thompson, G.L. "An Algorithm for Assigning uses to Sources in a special class of Transportation Problems," Operations Research, 21, 1973, pp.284-295.

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

[15] Venkata Ramana, V.V. & Umasankar, C. "On a Class of Assignment Problems," Opsearch, 1998, Vol.35, No.2. pp.127-138.

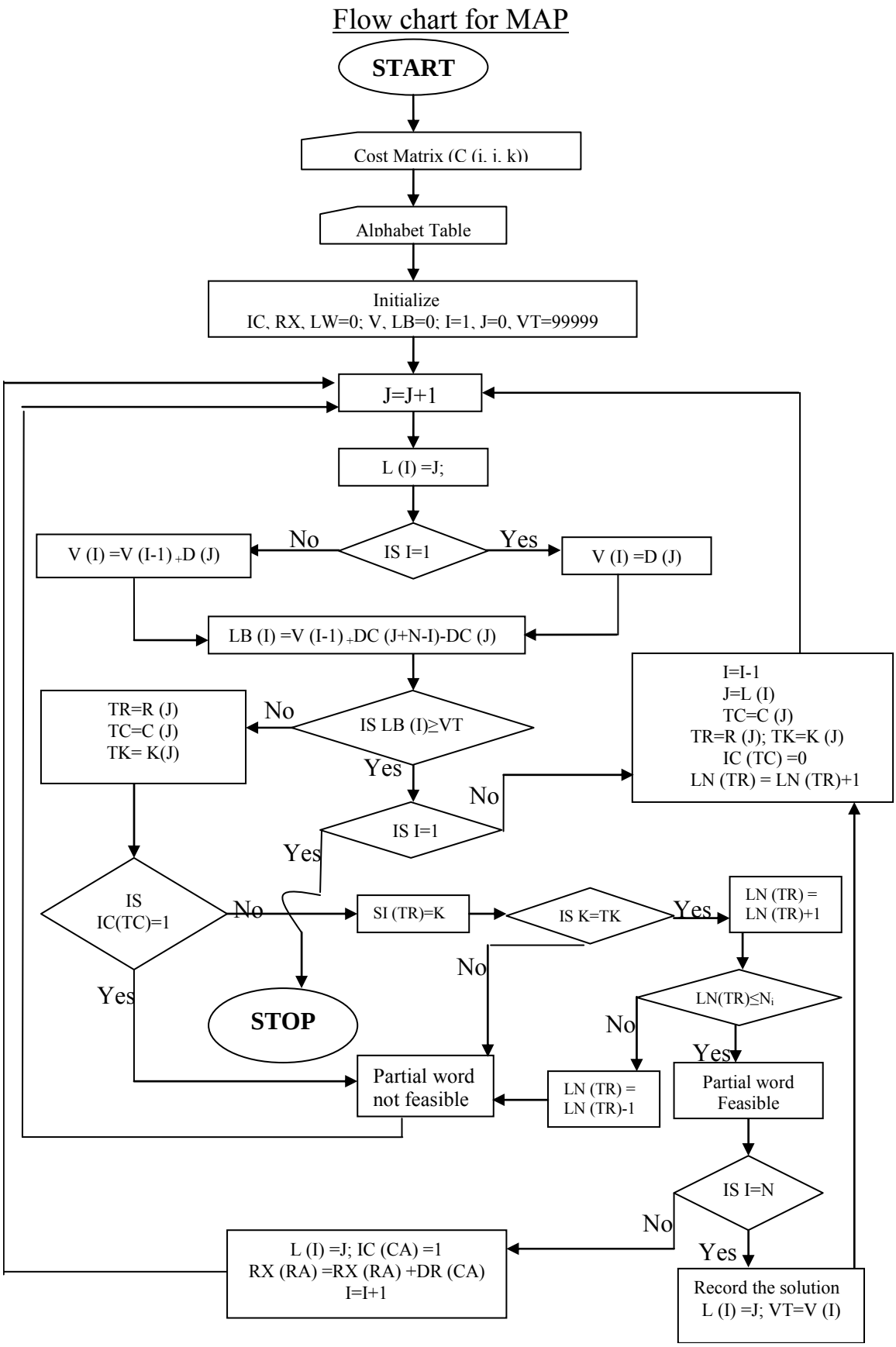
AUTHORS PROFILE

Mr.K.SOBHAN BABU is presently working as a Assistant Professor in the Department of Mathematics, University College of Engineering, JNTUK, KAKINADA, Andhra Pradesh, INDIA.

Mrs.K.CHANDRA KALA is presently working as a Assistant System Engineer in Tata Consultancy Services, Hyderabad, Andhra Pradesh, INDIA.

Mr.S.PURUSHOTTAM is a Research Scholar in the Department of Mathematics, Sri Venkateswara University, Tirupati, Chittor (Dt), Andhra Pradesh, INDIA.

Dr.M.SUNDARA MURTHY is presently working as a Professor in the Department of Mathematics, Sri Venkateswara University, Tirupati, Chittor (Dt) Andhra Pradesh, India.


Flow chart for MAP
START
Cost Matrix (C (i, j, k))
Alphabet Table
Initialize
IC, RX, LW=0; V, LB=0; I=1, J=0, VT=99999
J=J+1
L (I) =J;
IS I=1
No
Yes
V (I) =V (I-1) +D (J)
V (I) =D (J)
LB (I) =V (I-1) +DC (J+N-I)-DC (J)
IS LB (I)≥VT
No
Yes
TR=R (J)
TC=C (J)
TK= K(J)
IS I=1
No
Yes
I=I-1
J=L (I)
TC=C (J)
TR=R (J); TK=K (J)
IC (TC) =0
LN (TR) = LN (TR)+1
IS IC(TC)=1
No
Yes
SI (TR)=K
IS K=TK
Yes
No
LN (TR) = LN (TR)+1
LN(TR)≤Ni
No
Yes
STOP
Partial word not feasible
LN (TR) = LN (TR)-1
Partial word Feasible
IS I=N
No
Yes
L (I) =J; IC (CA) =1
RX (RA) =RX (RA) +DR (CA)
I=I+1
Record the solution
L (I) =J; VT=V (I)