IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728,p-ISSN: 2319-765X, Volume 6, Issue 4 (May. - Jun. 2013), PP 01-08
www.iosrjournals.org

Lexisearch Approach to Travelling Salesman Problem

G.Vijaya Lakshmi

Abstract: *The aim of this paper is to introduce Lexisearch the structure of the search algorithm does not require huge dynamic memory during execution. Mathematical programming is concerned with finding optimal solutions rather than obtaining good solutions. The Lexisearch derives its name from lexicography .This approach has been used to solve various combinatorial problems efficiently , The Assignment problem, The Travelling Salesman Problem , The job scheduling problem etc. In all these problems the lexicographic search was found to be more efficient than the Branch bound algorithms. This algorithm is deterministic and is always guaranteed to find an optimal solution.*

Keywords: *Introduction, Combinatorial problem , The Travelling Salesman problem, Lexisearch Method, The Lexisearch Approach, Algorithm LEXIG TSP, A Lexisearch Method Illustration of Travelling Salesman Problem, Conclusion.*

I. Introduction

Mathematical programming is concerned with finding optimal solutions rather than obtaining good solutions. The concept of optimization is ancient and has accelerated enormously with the development of computers and linear programming in the late 1940's. The mathematical programming problems are broadly classified into two classes namely:

1. Continuous programming problems
2. Discrete programming problems.

The problem of mathematical programming is to find the maximum or minimum of an objective function whose variables are required to satisfy a set of well-defined constraints.

If the objective function is continuous in the variable values that also lie in a feasible region, which is compact, then it is a continuous programming problem. This type of problems are solved by simplex procedure (Hadely 1994), Lagrangian multipliers method (Rao, 1984), where as if the decision variables take only discrete values then it is known as discrete programming problem.

If the set of solutions is a finite discrete set or not necessarily of variables in the usual sense but, May even be of other entities like permutations or combinations, then the problem is one of discrete programming.

Operations research problems are outlined slightly combinatorial and non-combinatorial having a common feature in that the objective function is to be minimized is the maximum of a set of function values.

Combinatorial problem: If the solution space of a problem consists of combinatorial entities, then it is known as combinatorial problem.

According to Pandit (1963), the combinatorial programming problems can be described as:
"There is a numerical function defined over the domain of arrangements or selections of a set of elements. There are also feasibility criteria .Now, the problems to find the arrangements which are feasible and which optimize the numerical function".

The combinatorial problems are not new to mathematics; perhaps one of the ways of characterizing them is by saying that they are concerned patterns, usually sequences that can be found out of a finite number of elements. the usual combinatorial problems are concerned with enumeration of patterns belonging to a given structure; class-considerable areas of graph theory are of this type.lke the mathematical problems in general, in combinatorial programming also we are not so much interested in the number of patterns of a given type, but in associating in the first place, a numerical value to every pattern of interest and then asking as to how to pick up that pattern which minimizes (or maximizes)this numerical value, among a well defined set(essentially of a finite cardinality) of patterns.

In a sense, "the range of the variable of the combinatorial functions" is the set of patterns-often permutations, combinations are more complicated symbol chains and the very convenient concepts of neighboring smoothness and related notions are completely irrelevant in this minimax optimization problem.

Some types of combinatorial programming problems have the markovian feature in the sense that they can be sequentially tackled, with the structure enabling that the section of solution subsequent to one stage of the solution is independent of the earlier part of solution but depends only on the current stage solution. It can be identified with associatively are the stages of solutions and when solutions of the problem can be formulated in

terms of an appropriate operation, it will be seen that the concerned operation is associative and is of great significance numerically. If not also theoretically, solving the problem. Many combinatorial programming problems unfortunately do not fall in this category they are non-markovian in that at any stage of solution, residual part of the problem cannot be linked from the partial solution up to that stage. A classical example of this type is the Travelling salesman and the Assignment problems. One has perhaps to be content with a predictable algorithm to solve with the problem though as in Assignment problem, the combinatorial structure may lead to interesting side results, which, if one is lucky, can also be of some combinatorial significant.

In fact, the area of combinatorial programming is full of such problems with some justification, we may theoretically say that the unifying factor which binds all these problems into one class is that they cannot be naturally brought under one head by any positive definition, and perhaps the only way of tackling is by devising suitable algorithms based on notions of hieratically structured numerical set of functions i.e., patterns, that cannot be arranged in a hierarchy in the value of the function which leads to the Lexisearch and branch bound algorithms.

Comparison of algorithms for solutions of any such problems is rather difficult and is often inevitable judgments about relative weight age to be given to different components of computation. It is customary to ignore the number of additions and subtractions but divisions in the execution of the algorithm. Combinatorial programming algorithms often involve large amounts of multiplication on the context of locating in the computer memory the various dimensional variables.

In the past one may be forced to accept the inevitable but highly unsatisfactory criticism of the time required for solving a number of problems with randomly generated data on any available computer, which was very slow. But the present scenario is quite different where we can generate a number of problems of various sizes and solve then with no time, because of the fast computing facility. A problem, which is essentially not combinatorial, is the one where algorithms for its solutions are developed in the analytic way using continuity concepts, but physically it appears that there should be an optimal solution to any arbitrary problem of this type. However, it is not always possible to recognize a proposed solutions, one can define a set of homogeneous equations with some variables restricted to be negative and ask whether the system has solution. If the answer is YES, the solution can be necessarily being improved. If the answer is NO it is an optimal solution. So, various authors answered this question one way or other by a few trials modifying the solution without modifying the minimax value and ultimately being able to get an identifiable optimal solution.

The Travelling Salesman Problem

A set of n nodes (cities), with distance between every ordered node pair, is given. The problem is to find a least distance route for a salesman who must visit each of these cities starting and ending his journey in the same city.

To solve the combinatorial problems different techniques are available. However, no general approach, which is suitable for all discrete programming problems, seems to be available and the methods of finding optimal solutions mainly by search methods (Pandit 1963). For many of the combinatorial programming problems the solution space is finite and hence, it is theoretically to enumerate all the solutions. But often, even for problems of small size, the number of solutions can become very large and grows exponentially, with the size of the problem making a complete enumerative solution not practicable. Hence there arises a need for implicit enumeration methods. In these methods through theoretically all the solutions are examined, only a few are explicitly to be examined. The efficiency of the implicit enumeration algorithms basically depends on how quickly it eliminates subsets of solutions as not including any optimal solutions and converges to an optimal feasible solution.

There are two methods in this implicit enumeration approach. They are:

1. Branch and bound algorithms(little et al , 1963)
2. Lexicographic search(Pandit 1963)

II. Lexisearch Method

The Lexisearch approach was first proposed by Pandit (1962) in the context of 'The Loading problem'. From 1963 onwards this approach has been used to solve various combinatorial problems efficiently, e.g., the Assignment problem (Jain, Mishra and Pandit, 1964). The faculty location problem (Das 1976), the Travelling Salesman Problem (Pandit and Rajbongshi, 1976; Sundara Murthy, 1979; Subramanyam, 1980; Ramesh 1980; Chandrasekhar Reddy, 1987). The job scheduling problem (gupta, 1967; Rajbongshi, 1982; Bhanumurthy, 1986). In all these problems the lexicographic search was found to be more efficient than the Branch bound algorithms (c.f.Ravi Kumar, 1989). It is viewed that in general sense, the Branch Bound approach can be viewed as a particular case of lexicographic search.

Travelling salesman problem (TSP, for short) is one of the oldest combinatorial programming problems as defined earlier ;(Flood, 1956, and Croes, 1958) can be stated as follows (c.f. sundara Murthy, 1979). The

objective of TSP is to find a minimal length tour i.e. a tour, which is of minimal length. The TSP can be defined equivalently as the problem of finding Hamiltonian cycle of the shortest length in a connected weighted graph.
Meril Flood who was associated with RAND Corporation gave wide publicity to the TSCP. The importance of the TSP comes out not from the wealth of applications but from the fact that it is typical of other problems of its genre: combinatorial optimization. Another reason for prominent topics in combinatorial problems arising in the new subject of LPP, The assignment problem and more generally, the transportation problem. TSP has some similarity with those other problems but is much harder to solve.

The combinatorial structure of this problem is very close to that of the assignment problem (AP). The only difference is that the solution set of the TSP is more restrictive, viz., the permutation matrix, say X, which is a feasible solution of the assignment problem will also be a feasible solution of the TSP only if it is non decomposable. This additional restriction brings some complexity into solving this problem and for the same reason; many of the methods, which solve the assignment problem, fail to do in this case.

The combinatorial structure of this problem is very close to that of the assignment problem (AP). The only difference is that the solution set of the TSP is more restrictive, viz., the permutation matrix, say X, which is a feasible solution of the assignment problem will also be a feasible solution of the TSP only if it is non decomposable. This additional restriction brings some complexity into solving this problem and for the same reason; many of the methods, which solve the assignment problem, fail to do so in this case.

Dantzig, Fulkerson and Johnson (1954) solved a sizeable TSP, formulating it as an LPP. The requirement of indecomposibility of the permutation matrix X can be checked through a set of linear constraints grows very fast as the size of the problem increases. Thus, theoretically it is possible to treat the TSP as a n LPP. However in practice, due to very large number of these special constraints, which are to be checked for indecomposibility, it is not practicable to solve the TSP by this approach, except when the size of the problem is relatively small. The cutting plane method was used to solve the above LPP with integer constraints and they also seem to have used the concept of branch and bound in some sense to solve the same (c.f. Lawler et al 1985).

It may be noted that Das (1976) applied the Lexisearch for TSP and further modifications were considered by Sundara Murthy (1979) and Srinivas (1990). Before this approach to TSP and its various versions are developed, a brief discussion of the TSP and various other approaches of the same are in order. Little et al, in 1963 for solving the TSP, developed the branch bound method. From then onwards it has been widely used to solve different problems in Operations research.

The solution space of TSP is a subset of the solution space of the assignment problem in that every permutation of order n is solution for the AP indecomposable permutations are acceptable as solutions of TSP. Hence, any procedure to solve the AP can be incorporating a procedure to check for cycles in the permutation proposed as a solution.

This fact is exploited in converting the Lexisearch algorithm for AP into one for TSP (cf. Pandit, 1964, 1965). Despite the fact that the travelling salesman model applies directly to very useful sounding solution, namely that of salesman wishing to minimize his travel distance, most of the reported applications are quite different; seemingly unrelated problems are solved by formulating them as instances of the TSP.
Some of the applications are listed below:
There are certain approaches for exact solution of TSP and they are

1. Integer programming approach.
2. Dynamic programming approach.
3. Branch and bound algorithm.
4. East man's algorithm.
5. Lexisearch method.

The Lexisearch Approach

The lexicographic search approach to the combinatorial optimization problem, as already pointed out, was developed by Pandit in the first instance for the Knapsack problem (Pandit, 1962) and has since been applied to many other combinatorial problems like Job scheduling (cf. Gupta, 1969) etc., in addition to the assignment problem and Travelling salesman problem (cf. Das, 1976).

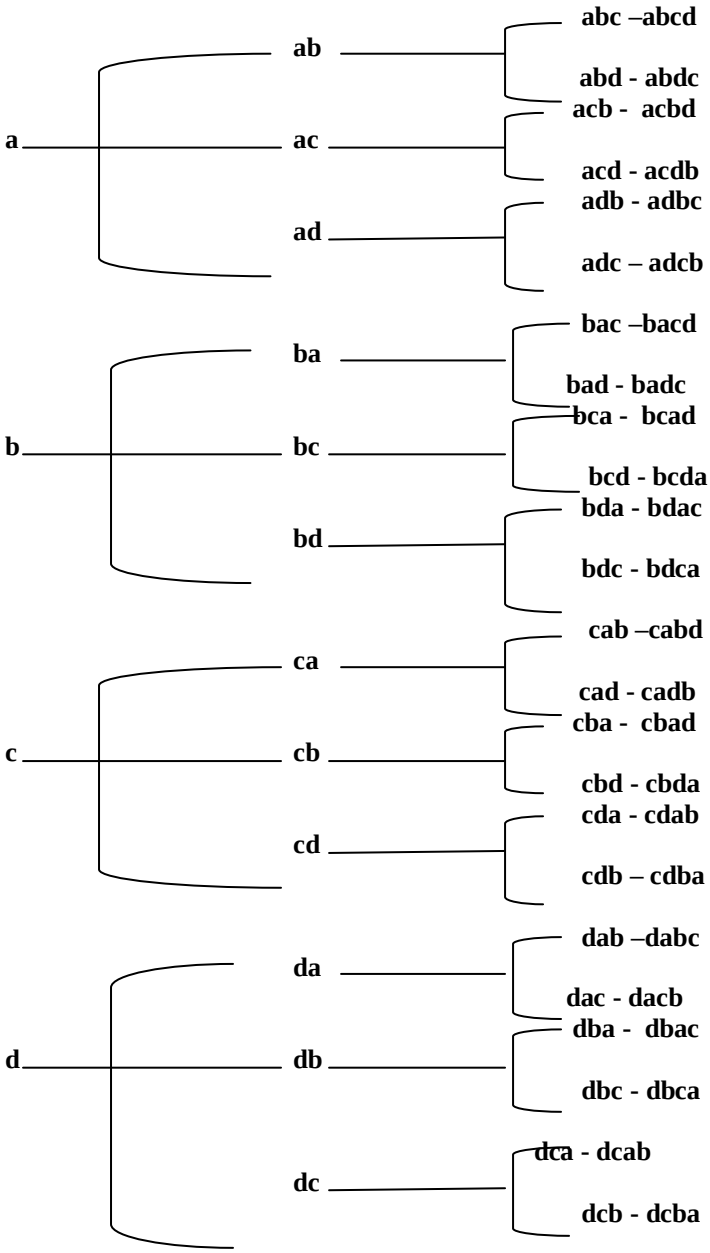
The Lexisearch derives its name from lexicography, the science of effective storage and retrieval of information. As already mentioned, the set of all possible solutions to a combinatorial programming problem is arranged in hierarchy-like words in a dictionary, such that each 'incomplete word' represents the block of words with this incomplete word as the 'leader'. Each node is considered as a letter in an alphabet and each tour can be represented as a word with this alphabet. Thus entire set of 'words' in this 'dictionary' (Viz. the set of solution) is portioned in to 'blocks'. A block 'B' with a 'leader ($\alpha 1$, $\alpha 2$, $\alpha 3$) of length three' consists of all the words beginning with ($\alpha 1$, $\alpha 2$, $\alpha 3$) as the string of first three letters.

The block 'A' with 'leader (α1, α2) of length' is the immediate 'super block' of B and includes B as one of its sub blocks. The block 'C' with leader (α1, α2, α3, β), one for each β_i. The block 'B' is the immediate super block of block 'C'. By the structure of the problem it is often possible to get bounds to the value of all words in a block (i.e., the 'bound for the block') by an examination of its leader. Hence by comparing that bound with the value of a trial solution. One can

1. Go into the 'sub blocks' if the block-bound is less than the trial solution value and hence, the current block may contain a solution better than the trial solution value.
2. Jump over to the 'next' block; if the block-bound is greater than the trial solution value, or
3. Jump out to the next 'super block'; if the current block, which is to be jumped over is the last block of the present super block.

Further, if the value of the current leader is already greater or equal to the current trial value; no need for checking the subsequent blocks within this super –block. These concepts are illustrated below; in case of the TSP.

Let a, b, c, d be the four cities to be travelled by a salesman. Then the set of possible partial and complete word is listed lexicographically as follows:



The words starting with 'a' constitute a 'block' with 'a' as its leader. In a block, there can be many sub-blocks; for instance 'b a', 'b c', and 'b d' are leaders of the sub-blocks of block b. There could be blocks with only one

word; for instance, the block with leader 'abd' has only one word 'abdc'. All the incomplete words can be used s leaders to define blocks. For each of blocks with leader 'ab','ac' and 'ad; the block with leader 'a' is the immediate super - block.

The following notations are used in this algorithm

W: current/partial solution

TR: trial value associated with a trial solution.

$1 \rightarrow \alpha_{1,1} \rightarrow \alpha_{1,2} \rightarrow \cdots \rightarrow \alpha_{1,n1} \rightarrow 1 \rightarrow \alpha_{2,2} \rightarrow \cdots \rightarrow \alpha_{2,n2} \rightarrow 1 \ldots 1 \rightarrow \alpha_{k,1} \rightarrow \alpha_{k,2} \rightarrow \cdots \rightarrow \alpha_{k,n} \rightarrow 1$

α_{ij} : The city visited by the i^{th} salesman as j^{th} city in his tour.

n_i: The number of cities visited by i^{th} salesman i=1, 2 ... k and n_i=N-1.

Algorithm LEXIG TSP

Step1: Arrange all cities according to their distances from the city i =1, 2… N+ (k-1). This arrangement consists of N+ (k-1) columns and N-1 rows since the origins are removed. Each column represents a city 'A' and the elements in that column are the cities arranged in increasing order according to their distance from the city 'A'.

Step 2: Include the first reachable city from the origin in the partial solution 'W'. If the distance itself is greater than or equal to TR then stop. Otherwise go to next step.

Step 3: calculate the Bound.

Step 4: if the (Bound+ Partial solution value) is greater than or equal to trial solution then drop the city added in step 2. Now go to step 2 i.e. JB otherwise go to next step i.e. GS.

Step 5: Include the next reachable city (from the last city included in the partial solution 'W') into the partial solution.

Step 6: If partial solution value is greater than or equal to the TR, drop the city added in step 5. Also drop the last city in 'W'. Go to step 5(i.e. JO).

Otherwise goes to step 7.

Step 7: If the (Partial solution value + Bound) is greater than or equal to TR, then drop the newly added city in step 5. (I.e. JB). Go to step 5.

Otherwise go to step 8.

Step 8: If the partial solution contains $\sum n_1$+ (i-1) cities, add the dummy origin to the partial solution. Calculate the value of this partial solution. If it is greater than or equal to the TR, drop the dummy origin and last two cities from the partial solution (i.e. JO).

If 'W' contains only one city, go to step 2; otherwise go to step 5

Otherwise go to next step.

Step 9: Calculate the Bound.

Step 10: If the (Partial solution value + bound) is greater than or equal to TR then drop the dummy origin and also last city from 'W'. Go to Step 5 (i.e., JB).

Otherwise go to step 11.

Step 11: Include the latest possible city from the dummy origin i in 'W'.

Step 12: If (Partial Solution value) is greater than or equal in TR, drop the last dummy origin and also the city from which the i^{th} dummy origin was reached. Go to step 5.

Otherwise go to next step.

Step 13: Calculate the Bound.

Step 14: If (Partial Solution value + Bound) is greater than or equal to TR, drop the recently added city in 'W' and go to step 11. Otherwise go to next step.

Step 15: Include the latest reachable city from the last city W.

Step 16: If (partial solution value) is greater than or equal to TR, drop the last two cities; If dummy origin happens to be one of these two cities, also drop the city from which the dummy origin was reached, reduce the value of i by I i.e. 'i' becomes i-I. Otherwise go to next step.

Step 17: Calculate the Bound.

Step 18: If (Partial Solution value + Bound) is greater than or equal to TR, drop the latest city i.e., JB. Go to step 15.

Otherwise go to next step.

Step 19: If the number of elements in W is less than ($\sum n_i$+k) go to step 15.

Otherwise go to step 20.

Step 20: Add the dummy origin to W and calculate the value of W. If it is greater than or equal to the TR drop the dummy origin and also the last two cities in W.

Go to step 15. Otherwise go the step 21.

Step 21: Replace TR by Partial Solution value and trial solution by W. Drop the dummy origin and last two cities in W. Go to step 15.

The symbols used in table **III** are listed below:
GS: Go to sub-block, i.e., attach the first 'free' letter to the current leader.
GS for 'db' leads to 'dba' as augmented leader.
JB or GNSB: Jump the block i.e. go to the next block of the same order i.e.,
Replace the last letter of the current block by the letter next to it in the alphabet table. JB or GNSB (GO TO NEXT SUPER BLOCK) for 'abc' 'adb'.
JO: Jump out to the next, higher order block, i.e., drop out the last letter of the current letter and then jump the block.
JO for 'cdbe' is 'cde'.
TRVL: Currently trial solution value.
CR: Column repletion.
Vt: Trial Value.

A Lexisearch Method Illustration of Travelling Salesman Problem.
Table I
Consider a 4 x 4 city problem

	1	2	3	4
1	∞	41	68	38
2	5	∞	97	46
3	39	38	∞	4
4	42	91	75	∞

Alphabet Table

Table II

1	2	3	4
4 - 38	1 - 5	4 - 4	1 - 42
2 - 41	4 - 46	2 - 38	3 - 75
3 - 68	3 - 97	1 - 39	2 - 91
1 - ∞	2 - ∞	3 - ∞	4 - ∞

Lexisearch Search Table

Table III

1	2	3	4	Bound	A.B	Remarks
4 →38	-	-	-	5 + 38 + 42=85	38 + 85 =123<Vt	GS
	1→ 5 (38 + 5=43)	-	-	38 + 75=113	113+43=156<Vt	GS
		4-CR 2 → 38 (43 + 38 = 81)	-	75	81+75=156<vt	GS
			3 → 75 = 156	0	Vt=156	JB
		1-39 CR 3-∞				Go to next super block
	4-CR 3→ 97 (38+97=135)	-	-	80	80+135=215>vt	JB
	2→ ∞	-	-	-	-	GNSB
2 → 41(41)	1-is a cycle 4 → 46(41 + 46 = *87*	-	-	81	87+81=168>vt	GS
	3 → 97(41 + 97 = *138*	-	-	46	138+46=184>vt	JB
	2→ ∞	-	-	-	-	GNSB
3→ 68	-	-	-	51	68+51=119<vt	GS
	1→ 5 (68+5=73)	-	-	95	168>vt	JB

	4→ 46 (46+68=114)	-	-	80	194>vt	JB
	3-CR	-	-	-	-	
	2→ ∞	-	-	-	-	GNSB
1→ ∞						

Let Vt = ∞, be a Trial Value

The first tour that naturally occurs to our mind viz., $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$ *is* taken as the trial solution with the value 156 i.e., 38 + 75 + 38 +5 = 156. Here 156 < Vt. Now, to find the optimum solution, one start from origin 1, that is , node '1' and reaching in the order of first available nodes in the appropriate columns of alphabet table and returning to origins 1.

Thus for instance, the search starts from column 1 whose first available node is 4, which is minimum in cost, and its cumulative value is 38. The bound is obtained by taking the first available nodes of appropriate columns and in this we observe that, the node first we have taken should not be reached. Thus the bound for the link $1 \rightarrow 4$ *is* $5 + 38 + 42 = 85$ (in column 2, the first available node is '1' in column 3, the first available m=node is 4, which is repeated, hence we should not consider and we go for second node in the same column i.e. 2, which is not repeated, whose value is 38. In column 4, the first available node is '1' which is not repeated and having the value 42).

Now, for this additive bound is 38 + (5+38+42)=38+85=123<Vt hence go to sub block (GS) i.e., $2 \rightarrow 1$ *is* 5. For this, the bound is 38 + 75 = 113. It is obtained by talking the first available node values, from the columns 3 & 4..

Here, we should observe that node 4, 1 are taken, white calculating the bound for link $2 \rightarrow 1$ except 4, 1 we can take the nodes 3 or 4 which are the first available nodes in columns 3 & 4.

In column 3, the first available node is 4, which is repeated, then go for next node in the same column i.e., 2 whose value is 38. In column 4, the first available node is '1' which repeated then go for next node in the same column, which is 3 and its value is 75.

The additive bound or this link $2 \rightarrow 1$ is 43 + (38 + 75) = 43 + 113 =156< Vt.

Then go to sub block i.e., 4, which is repeated column repetition (CR).

In the same column 3, the next node is 2 whose value is 38. The bound and additive bound for this link $3 \rightarrow 2$ is calculated as explained above.

The additive bound for this link $3 \rightarrow 2$ is 81 + 75 = 156.

Then go to sub block i.e., $4 \rightarrow 3$ (75) whose bound and additive bound is calculated as explained above. i.e. 156.

Consider this, 156 as Trial Solution Value. Then jump to next block i.e., $3 \rightarrow 1$ which is repeated (CR) then next available node is $3 \rightarrow 3$ is ∞. Then go to next super block (GNSB).

i.e., $2 \rightarrow 4$ which is repeated (CR) in the same column the next available node is 3 whose value is 97 and the additive bound is 215 which is greater than TRV. The jump block, the next link is $2 \rightarrow 2$ is ∞. Then go to next super block. Like this if we proceed is this way to get optional solution.

For, the above search table the optimal solution is 156, and then the complete word which belongs to the sequence is $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

III. Conclusion

In the Lexisearch the structure of the search algorithm does not require huge dynamic memory during execution. Hence, particularly for larger problems Lexisearch appears to have relatively smaller space complexity. Also, Lexisearch allows parallelization of the algorithm as compared to the Branch and Bound algorithm. In terms of the complexity also Lexisearch appears quite competitive as reported by many investigators on the basis of simulation studies. Till recently, the approach of Lexisearch methodology is established in various fields of operations research and this method is being tried in parallel computing. This algorithm described a search based algorithm for finding optimal solution to the Travelling Salesperson Problem. This algorithm is deterministic and is always guaranteed to find an optimal solution, unlike the conventional dynamic programming or Branch and Bound algorithms, which require exponential space; the lexicographic algorithm required only a linear space with respect to the problem size. This algorithm is easily parallelizable and found that this method is superior to the existing methods.

References

[1]. M. Ramesh. *"A lexisearch approach to some combinatorial programming problems"*, University of Hyderabad, India, 1997.
[2]. S.N.N. Pandit. "*Some quantitative combinatorial search problems*", Indian Institute of Technology, Kharagpur, India, 1963.
[3]. Srinivasan V. & G.L. Thompson (1973). An Algorithm for Assigning Uses to Sources in a Special Class of Transportation Problems, Op. Res., Vol. 21, No.1.

[4]. Subramanyam, Y. (1980): Scheduling, Transportation & Allied Combinatorial Programming Problems, Ph.D. thesis, REC Warangal, India (Unpublished).
[5]. Kanthi Swarup ,P.K.Guptha ,Man Mohan-"*Operations Research*"- Sulthan Chand and Sons 13th Edition.
[6]. Hamdy A.Taha-"*Operations Research-An Introduction*"-(7th edition),Pearson Education (Singapore).
[7]. J.K .Sharma -"*Operations Research-Theory and applications*"-MacMillan India Ltd.
[8]. S.C. Sharma –"*Introductory operations Research*"- e books.
[9]. P.Rama Murthy-" *Operations Research linear programming"*-e books.