A Lexisearch Algorithm for the Distance-constrained Vehicle Routing Problem

ZAKIR HUSSAIN AHMED
Department of Computer Science
Al Imam Mohammad Ibn Saud Islamic University
P.O. Box No. 5701, Riyadh-11432
KINGDOM OF SAUDI ARABIA
E-mail: zhahmed@ccis.imamu.edu.sa

Abstract: - The vehicle routing problem (VRP), belongs to the class of *NP*-hard problems, is considered as one of the most difficult problems. The problem aims to serve a number of customers with a fleet of vehicles that can be defined as to find an optimal set of tours with minimum cost to connect the depot to *n* customers with *m* vehicles, such that every customer is visited exactly once; every vehicle starts and ends its tour at the depot. Out of many variations of the problem, we consider here distance-constrained VRP (DVRP) where the total travelled distance by each vehicle in the solution is less than or equal to the maximum possible travelled distance. The problem has many applications in real-life and no any polynomial time exact algorithm is available to solve the problem, and even small sized instances may require long computation time. However, there are some situations where exact solution is required. Therefore, we look for exact solution to the problem, and accordingly, we develop a lexisearch algorithm to obtain exact solution to the problem, and present solution for the DVRP using some TSPLIB instances of various sizes.

Key-Words: - The vehicle routing problem, distance-constrained, exact algorithm, lexisearch.

1 Introduction

The vehicle routing problem (VRP) can be defined as a problem of finding the optimal routes of delivery or collection from one or several depots to a number of cities or customers, while satisfying some constraints. The objective is to minimize the cost (time or distance) for all tours. The cost of the tours can be fuel cost, driver wages, and so on. It is an NP hard problem [1]. The classical VRP can be defined as follows [2]:

Let G = (V, A) be a graph, where V = {1, 2, …., n} is a set of vertices (nodes) representing cities with the depot located at vertex 1, and A is the set of arcs with every arc (i, j), $i \neq j$, associated a non-negative distance matrix D = (d_{ij}). In some contexts, d_{ij} can be interpreted as a travel cost or travel time. When D is symmetric, it is often convenient to replace A by a set E of undirected edges. In addition, assume that there are m available vehicles based at the depot, where $m_L < m < m_U$. When $m_L = m_U$, m is said to be fixed. When $m_L = 1$ and $m_U = n - 1$, m is said to be free. When m is not fixed, it often makes sense to associate a fixed cost f on the use of a vehicle. The VRP consists of designing a set of least-cost vehicle routes in such a way that:

(i) each city in V \ {1} is visited exactly once by exactly one vehicle;
(ii) all vehicle routes start and end at the depot;
(iii) some side constraints are satisfied.

Collection of household waste, gasoline delivery trucks, goods distribution, street cleaning, distribution of commodities, design telecommunication, transportation networks, school bus routing, dial-a-ride systems, transportation of handicapped persons, and routing of sales people and maintenance units, snow plough and mail delivery are the most used applications of the VRP. The VRP plays a vital role in distribution and logistics. Huge research efforts have been devoted to studying the VRP since 1959 where Dantzig and Ramser [3] have described the problem as a generalized problem of multiple travelling salesman problem (m-TSP), which is also an NP-hard problem. Lenstra and Rinnooy Kan [4] suggested transforming VRP into *m*-TSP by adding (*m* - 1) dummy vertices (where *m* is the number of vehicles). The number of vehicles in the VRP

corresponds to the number of salesmen. There are many variations of VRPs: based on data (static and dynamic); based on constraints (constrained and unconstrained) (see [2, 5-7]).

The distance-constrained vehicle routing problem (DVRP) is defined as follows: find the optimal set of tours to connect the depot to **n** customers with **m** vehicles, such that:

(i) Every customer is served exactly once.
(ii) Every vehicle starts and ends its tour at the depot.
(iii) The total distance travelled by each vehicle in the solution is less than or equal to the maximum distance allowed (D_{max}).

If we have only one vehicle with no distance constraint, the DVRP will be equivalent to the TSP. The solution in this case entails looking for one tour over all customers with minimum travelled distance.

The aim and objective of this paper is to develop an exact algorithm to solve the DVRP. Since, the problem is NP-hard, it is very difficult to solve, and in fact, till the date no exact algorithm has been found, which is able to solve the problem in polynomial time. Finding an exact solution to the problem is usually very time consuming or even impossible. Because of this nature of the problem, a very few literatures on exact methods have been used to solve the instances of the problem. However, there are some situations where only exact solutions to the instances of the problem are required. Hence, we go for exact solution of the problem.

This paper is organized as follows: Section 2 presents literature review on the problem. Section 3 presents a formulation of the problem. A lexisearch algorithm is developed in Section 4. Computational experience for the algorithm has been reported in Section 5. Finally, Section 6 presents comments and concluding remarks.

2 Literature review

The VRP belongs to the class of NP-hard problems and is considered as one of the most difficult problems. Because of the extreme difficulty of the problem, exact solution methods have been often implemented on high-performance computers. Methods to solve the VRP as well as any combinatorial optimization problem are classified into two broad categories – exact and heuristic. Exact methods give exact solution to the problem. An implicit way of solving the VRP is simply to list all the feasible solutions, evaluate their objective function values, and pick out the best. However it is obvious that this 'brute-force technique' is grossly inefficient and impracticable because of vast number of possible solutions to the problem, even for problem of moderate size. In fact, computational time grows exponentially with the problem size. For example, we are listing our n customers in some order (which can be done in exactly n! ways), and we then place m –1 delimiters that determine when a route has ended after m− 1 out of the n − 1 (placing it after the last customer creates an empty vehicle) customers, which can be done in exactly $\binom{n-1}{m-1}$ ways, creating a total of $\frac{n!\binom{n-1}{m-1}}{m!}$possible solutions (we divide by m! because the order of the vehicles is irrelevant). One can imagine that with more than 10 customers and 3 vehicles, this method will soon be way too complex. Even though there are smarter ways of implementing a solver for the VRP with a brute-force technique, because the problem is NP-hard, it is highly unlikely that we will ever find the exact solution with a brute-force technique. We will have to try to be a little smarter.

Quite a few special exact algorithms have been developed, which can solve the problem much more efficiently than brute-force technique. However, it is observed that as the problem size increases obtaining exact solution to the problem is very difficult. On the other hand, heuristic methods don't guarantee the optimality of the solution, but give near exact solution very quickly. However, there are situations where exact solutions are very important.

In this paper, we consider the distance-constrained VRP (DVRP) where the total travelled distance by each vehicle in the solution is less than or equal to the maximum possible travelled distance. If the distance from city i to city j differs from that of city j to city i, we call this problem asymmetric (ADVRP), otherwise it is called as symmetric (SDVRP). We seek exact solution to the DVRP of both cases.

The different methods used to achieve a global optimum for the VRP and its variations include branch-and-bound, cutting planes or combinations of these methods, like branch-and-cut and dynamic programming. Branch-and-bounds are the most known and used algorithms and are defined from allocation and cutting rules, which define lower bounds for the problem .

A branch-and-bound algorithm for the VRP clearly requires a lower bound, because we have to minimize the total cost. Over the past 50 years, many lower bounds have been suggested for the VRP. An excellent survey of lower bounds is given in [8]. In Fisher [9], a description of a branch-and-bound algorithm for the VRP is given. This algorithm converts the VRP into a so-called "K-

tree", a structure for which a polynomial algorithm exists to find shortest paths. Amongst other smart things, this algorithm partitions the problem by fixing the edges between certain clustered customers. Some side constraints that take care of the vehicle capacity and the fact that each customer is visited at most once are also added. This algorithm has produced proven optimal solutions for a number of difficult problems, including a well-known problem with 100 customers. However, it still leaves certain 50-customer problems unsolved .

Recent solution techniques are mostly based on branch-and-cut, branch-and-cut-and-price, or column generation [7, 10-12]. Though there is a rich literature for the VRP, however, there is very poor literature for the DVRP.

Laporte and Desrochers [13] developed two exact algorithms for DVRP - one is based on Gomory cutting planes and other one is based on branch and bound. They deal with symmetric Euclidean and non-Euclidean instances. They concluded that solving non-Euclidean instances is easier than solving Euclidean. Both of their algorithms are able to find the optimal solution with up to 50 customers in Euclidean and 60 in non-Euclidean instances. Also, it is observed that the cutting plane algorithm performs better than branch and bound algorithm. Furthermore, in both algorithms, solving instances become more difficult when the maximum distance allowed is decreased.

Laporte et al. [14] presented an integer linear programming algorithm to solve VRP with distance and capacity constraints. They use relaxation constraints and subtour elimination constraints. They could solve the model with up to 60 customers using Euclidean and non-Euclidean instances.

Laporte et al. [15] developed another exact algorithm for solving ADVRP by using the branch and bound method where the relaxation problem is the modified assignment problem. They extended the distance matrix based on the technique of Lenstra and Rinnooy [4] by adding (m - 1) dummy depots where m represents the number of vehicles. The solution is feasible to the problem if following two conditions are satisfied- (a) the solution contains m Hamiltonian circuits, and (b) the length for each of them is less than or equal to the maximum distance allowed. If an infeasible solution is obtained, the infeasible circuit is eliminated by adding a new constraint. This means the illegal subtour is eliminated by branching the infeasible subproblem into subproblems. First feasible solution is obtained by adapting Clarke and Wright's algorithm [16]. If it does not obtain a feasible solution then the upper bound (UB) is increased by setting $UB = m\ X\ D_{max}$, where D_{max} represents the maximum distance allowed. If the total length of a subtour is greater than D_{max}, then it is eliminated by excluding arcs. The algorithm is then applied on randomly generated instances, and reported the solution up to 100 customers.

Li et al. [17] considered two objective functions to DVRP - minimize total distance and minimize number of vehicles. The DVRP is transferred into a multiple traveling salesman problem with time windows (mTSPTW), where the time window constraint $[a_i, b_i]$ for any customer i means that it is not allowed to serve customer i before a_i or after b_i. In other words, the vehicle has to wait until time a_i to start before dealing with customer i. In order to enable the transformation, the distance constraint is used as a time window constraint $[0, D_{max}]$ for all customers, and another copy of the depot is added to the graph. The time window for the first depot is [0, 0] (departure depot), and the time window for the last depot is $[0, D_{max}]$ (arrival depot). It is solved using a column generation approach. They presented and analyzed the worst case performance for DVRP with a heuristic and provided results with up to 100 customers. The comparison includes the length of the initial tour and the value of the lower bound.

A new algorithm for solving ADVRP based on Branch and Bound (B&B) method has been proposed [18] and found good solutions. As in [15], the ADVRP is first transformed into the TSP. The lower bounds are obtained by relaxing subtour constraints and maximum distance constraint. Thus, the assignment problem (AP) is got and solved in each city of the search tree. The best-first-search strategy is used and tolerance based rules are adapted for branching. As reported, instances up to 1000 cities could be solved .

There are some heuristic algorithms based on simulated annealing, tabu search, genetic algorithms [6], variable neighborhood search [19-21]. However, we are not applying any heuristic algorithm.

3 Formulation of the problem

Let V = {1, 2 . . . , n} denote the set of customers (cities), where 1 is the index of the depot. A set of edges is denoted by E, $E = \{(i, j) \in VxV, i \neq j\}$. The matrix D is defined as $D=[d_{ij}]$, where d_{ij} presents the travelled distance from city i to city j. The number of vehicles and the maximum distance allowed are denoted by m and D_{max} respectively. The problem is to find an optimal set of tours with minimum distance to connect the depot to *n-1 citie*s

(customers) with *m* vehicles, such that every city is visited exactly once; every vehicle starts and ends its tour at the depot, and the total travelled distance by each vehicle in the solution is less than or equal to D_{max}. If these constraints are relaxed, the VRP reduces to a TSP if ***m*** = 1 and to a m-TSP if m>1. The VRP appears to be much more difficult to solve than the TSP or the m-TSP involving the same number of cities.

There are many formulations of the DVRP. We consider the following mathematical programming formulations of the problem. This formulation is based on transformation of DVRP to TSP. We use its relaxation in our lexisearch algorithm.

The TSP formulation may be obtained by adding m - 1 copies of the depot to V [4]. Now, there are n + m - 1 cities in the new augmented directed graph G(V, A), where V = V U {n + 1, n + 2, . . . , n + m – 1}. The new distance matrix D' = [d'$_{ij}$] is obtained from given D by the following transformation rules where i,j ∈ V:

$$d'_{ij} = \begin{cases} d_{ij} & if\ \ (1 \le i,j \le n; i \ne j) \\ d_{1j} & if\ (i > n,\ \ 1 < j \le n) \\ d_{i1} & if\ (1 \le i \le n, j \ge n) \\ \infty \end{cases}$$

Then the formulation of TSP [22] given below (1)–(4) is as follows:

$$f(S) = min \sum_{(i,j)\in A} d'_{ij}\, x_{ij} \qquad (1)$$

where x_{ij} satisfies these conditions

$$\sum_{i\in V} x_{ij} = 1 \quad \forall\, j \in V \qquad (2)$$

$$\sum_{j\in V} x_{ij} = 1 \quad \forall\, i \in V \qquad (3)$$

$$\sum_{i\in V} x_{ij} \le |U| - 1 \quad \forall U \in V \setminus\{1\}, |U| \ge 2 \qquad (4)$$

Distance constraints (5)

The constraints (2) and (3) ensure that in and out degrees of each city are equal to 1. The constraint (4) eliminates subtours, where U is any subset of V\{1}. To formulate DVRP in addition to (1)–(4), we need to add distance constraint (5) to check if the total distance for each tour is less than maximum distance allowed (D_{max}).

For an instance of six-city and two-vehicle problem with distance matrix given in Table 1, the augmented distance matrix, constructed by adding 1 copy of the depot (city 1) row and column (i.e., 1st row and 1st column) to the given matrix, is reported in Table 2.

Table **1**: The distance matrix

City	1	2	3	4	5	6
1	999	2	11	10	8	7
2	6	999	1	8	8	4
3	5	12	999	11	8	12
4	11	9	10	999	1	9
5	11	11	9	4	999	2
6	12	8	5	2	11	999

Table **2**: The augmented distance matrix

City	1	2	3	4	5	6	7
1	999	2	11	10	8	7	999
2	6	999	1	8	8	4	6
3	5	12	999	11	8	12	5
4	11	9	10	999	1	9	11
5	11	11	9	4	999	2	11
6	12	8	5	2	11	999	12
7	999	2	11	10	8	7	999

4 A lexisearch algorithm for the problem

In lexisearch algorithm, the set of all possible solutions to a problem is arranged in a hierarchy, such that each incomplete word represents the block of words with this incomplete word as the leader. For the VRP, each node is considered as a letter in an alphabet and tour set can be represented as a word with this alphabet. Thus the entire set of words in this dictionary (namely, the set of solutions) is partitioned into blocks. Bounds are computed for the values of the objective function over the blocks of words, which are then compared with the 'best solution value' found so far. If no word in the block can be better than the 'best solution value' found so far, jump over the block to the next one. If the current block, which is to be jumped over, is the last block of the present super-block, then jump out to the next super-block. Further, if the value of the current leader is already greater than or equal to the 'best solution value'; no need for checking the subsequent blocks within this super-block. However, if the bound indicates a possibility of better solutions in the block, enter into the sub block by concatenating the present leader with appropriate letter and set a bound for the new sub-block so obtained [23-27].

4.1 Alphabet table

Alphabet matrix, A=[a(i,j)], is a matrix of order nX(n+m-1) formed by positions of elements of the augmented distance matrix, D=[d_{ij}]. The i^{th} row of matrix A consists of the positions of the elements in the i^{th} row of the matrix D when they are arranged in the non-decreasing order of their values. If a(i,p) stands for the p^{th} element in the i^{th} row of A, then a(i,1) corresponds to the position of smallest element in i^{th} row of the matrix D [23]. Alphabet table "$[a(i,j)-d_{i,a(i,j)}]$" is the combination of elements of matrix A and their values as show in Table 3.

Table **3**: The alphabet table (C is a city and W is the value of the city)

City	C-W	C-W	C-W	C-W	C-W	C-W	C-W
1	2-2	6-7	5-8	4-10	3-11	1-999	7-999
2	3-1	6-4	1-6	7-6	4-8	5-8	2-999
3	1-5	7-5	5-8	4-11	2-12	6-12	3-999
4	5-1	2-9	6-9	3-10	1-11	7-11	4-999
5	6-2	4-4	3-9	1-11	2-11	7-11	5-999
6	4-2	3-5	2-8	5-11	1-12	7-12	6-999
7	2-2	6-7	5-8	4-10	3-11	1-999	7-999

4.2 Incomplete word and block

An incomplete route (α_1, α_2,….., α_k) of k cities, where k≤n, is called an incomplete word or a leader of length k. This incomplete word is also called a leader of length k. A block Q with a leader (α_1, α_2, α_3) of length three consists of all the words beginning with (α_1, α_2, α_3) as the string of first three letters. The block P with leader (α_1, α_2) of length 2 is the immediate super-block of Q and includes Q as one of its sub-blocks. The block R with leader (α_1, α_2, α_3, α_4) is a sub-block of Q. The block Q consists of many sub-blocks (α_1, α_2, α_3, α_r) one for each α_r. The block Q is the immediate super-block of block R [23].

4.3 Lower bound

In lexisearch algorithm [23], solution does not depend on lower bound, unlike branch-and-bound algorithm. The lower bound for each block leader on the objective function value is set to skip as many subproblems in the search procedure as possible. A subproblem is skipped if its lower bound exceeds the 'best solution value' found so far (i.e., upper bound) in the process. The higher the lower bound the larger the set of subproblems that are skipped. Following method is used for setting lower bound for each leader.

Suppose the partial tour is (1=α_1, α_2, α_3) and 'city α_4' is selected for concatenation. Before concatenation, we check bound for the leader (α_1, α_2, α_3, α_4). For that, we start our computation from 2^{nd} row of the 'alphabet table' and traverse up to the n^{th} row, and sum up the values of the first 'legitimate' city (the city which is not present in the tour), including 'city 1', in each row, excluding α_2-th and α_3-th rows. This sum is the lower bound for the leader (α_1, α_2, α_3, α_4).

4.4 The algorithm

Step 0: Suppose, n be the number of cities including depot, m be the number of vehicles, distance travelled by the vehicles are Dist[i] for 1≤i≤m, 'city 1' be the depot, and D_{max} be the maximum distance allowed by each vehicle. Set 'best solution value (BS)' as large as possible, Dist[i]=0 for 1≤i≤m. Since 'city 1' is the starting city, we start our computation from 1^{st} row of the 'alphabet table'. Initialize 'partial tour value (PT)'=0, k=1 and go to step 1.

Step 1: - Go to k^{th} element of the row (say, city p) with value as present city value (W). If (PT+W)≥ (BS or D_{max}), go to step 7, *else*, go to step 2.

Step 2: - If all the vertices are visited, add an edge connecting the 'city p' to 'city 1', compute the complete tour value (CT) and go to step 3, *else* go to step 4.

Step 3: - If (CT≥BS), go to step 7, *else*, BS=CT and go to step 7.

Step 4: - Calculate the lower bound (LB) for the present leader on the objective function value and go to *step 5*.

Step 5:- If (LB+PT+W)≥BS, drop the 'city p', increment k by 1, and go to step 6; *else*, accept the 'city p', compute PT=PT+W, update Dist[k] for the present vehicle k, and go to step 1.

Step 6: - For any vehicle k, if (Dist[k]<D_{max}) go to step 1, *else*, go to step 7.

Step 7: - Jump this block, i.e., drop the present city, go back to the previous city in the tour (say, city q), i.e., go to the q^{th} row of the 'alphabet table' and increment k by 1, where p was the index of the last 'checked' city in that row. If vertex q = 1 and k = n, go to step 8, *else*, go to step 1.

Step 8: Now BS is the optimal solution value and calculate the maximum distance travelled by any vehicle (Max), and then *stop*.

4.3 Illustration of the algorithm

Working of the above algorithm is explained through a six-city and two-vehicle example with distance matrix given in Table 1. We obtain augmented distance matrix by adding 1 copy of the depot (city 1) row and column (i.e., 1st row and 1st column) to the given matrix, which is reported in Table 2. We will consider two problems with this dataset, i.e., they will have two different values of the parameter D_{max}.

The logic-flow of the algorithm at various stages is indicated in a search table that sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps. The symbols used therein are listed below:

GS: Go to sub-block.

JB: Jump the current block.

JO: Jump out to the next, higher order block.

Problem 1: First, the value of D_{max} is set to very large number, say, 999.

Illustration of the example: We initialize BS = 999 and PT=0, and Dist[1]=Dist[2]=0. We start from 1st row of the 'alphabet table'. Here, a(1,1)=2 with $W=d_{12}=2$, and (PT+W<BS) and (Dist[1]+W<D_{max}). Now we go for bound calculation for the present leader (1, 2). The bound will guide us whether the city 2 will be accepted or not.

$$LB = d_{2,a(2,1)} + d_{3,a(3,2)} + d_{4,a(4,1)} + d_{5,a(5,1)} + d_{6,a(6,1)} + d_{7,a(7,2)}$$
$$= d_{2,3} + d_{3,7} + d_{4,5} + d_{5,6} + d_{6,4} + d_{7,6}$$
$$= 1 + 5 + 1 + 2 + 2 + 7 = 18$$

Note that for calculating $d_{2,a(2,1)}$, we first find a(2,1) which is 3, then we find $d_{2,3}$ which is 1, and accordingly we calculate LB. Now, since (LB+PT+W=18+0+2=20< BS), we accept the city 2 that leads to the partial tour {1→2} with PT=PT+W=0+2=2, Dist[1]=2. Note that in the search table, $\begin{matrix}1 \to 2\ (2)\\ (0) + 18, GS\end{matrix}$ means W=2, PT=0, LB=18 and remark is GS.

Next, we go to 2th row of the 'alphabet table'. Since a(2,1) = 3 with $W=d_{23}=1$, and (PT+W<BS) and (Dist[1]+W< D_{max}), we go for bound calculation for the present leader (1, 2, 3).

$$LB = d_{3,a(3,2)} + d_{4,a(4,1)} + d_{5,a(5,1)} + d_{6,a(6,1)} + d_{7,a(7,2)}$$
$$= d_{3,7} + d_{4,5} + d_{5,6} + d_{6,4} + d_{7,6}$$
$$= 5 + 1 + 2 + 2 + 7 = 17$$

Since, (LB+PT+W=17+2+1=20 < BS), we accept the city 3 that leads to the partial tour {1→2→3} with PT=PT+W=2+1=3. Proceeding in this way we obtain the complete tour for the first vehicle as {1→2→3→7} with Dist[1]=8, and the tour for the second vehicle is {7→6→4→5→1} with Dist[2]=21. So the complete 1st tour is {1→2→3→7→6→4→5→1} and BS = 29 and Max=21. Now, we jump out to the next higher order block, i.e., {1→2→3→7→6→4}, and try to compute another complete tour with lesser tour value. Finally, we obtain the optimal tour as {1→2→3→7→6→4→5→1} with BS = 29 and Max=21. Optimal tours for the vehicles are {1→2→3→1} and {1→6→4→5→1} with their distances 8 and 21 respectively, and Max=21. The logic-flow of the algorithm at various stages is indicated in Table 4.

Problem 2: The longest tour in the optimal solution of Problem 1 is 21. The new value of D_{max} is calculated as D_{max} = 0.97x21=20.37 with respect to the matrix. We choose D_{max} = 20.37 and run the same example. Then our algorithm obtains the optimal tour as {1→2→6→4→5→7→3→1} with BS = 36 and Max=20. Optimal tours for the vehicles are {1→2→6→4→5→1} and {1→3→1} with their distances 20 and 16 respectively, and Max=20.

5 Computational experience

The lexisearch algorithm (LSA) has been encoded in Visual C++ on a PC with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz and 8.00GB RAM under MS Windows 7. We have selected some TSPLIB [28] instances of size from 17 to 45 with optimal solution value are reported within brackets for the usual TSP. We report the solutions (Max and Total) that are obtained within three hours (CTime, in seconds) on the machine as well as computational time when the final best solution is seen for the first time (FTime) for different number of vehicles (m) and different values of D_{max}. First value of D_{max} is ∞ is used to find Max(1), whereas second value of D_{max} is set as 0.9*Max(1) to find Max(2) and new solution value.

Table 5 reports results on asymmetric TSPLIB instances, whereas Table 6 reports results on symmetric TSPLIB instances. It is to be noted that no modification of the algorithm is required to solve different kind of problem instances. It is shown that whenever number of vehicle is one, the problem becomes usual TSP. As the number of vehicles increases the computational time as well as the solution value also increases, and the problem is found to be more difficult. Also, for the same problem instance, as the value of D_{max} decreases problem becomes more difficult. However, to the

best of our knowledge, no literature report optimal solution for the instances, and hence, we could not report comparative study. Except for the instances reported in the table, LSA could not prove the optimality of the solution of the instances within three hours of computational time.

6 Conclusion

We presented lexisearch algorithm for the distance-constrained vehicle routing problem to obtain exact optimal solution to the problem. We report best/optimal solution value and total computational time as well as the computational time when the final best solution is seen for the first time on the TSPLIB instances of various types. Of course, we did no modification in the algorithm for applying on different types of instances. It is seen that as the number of vehicles increases the computational time and optimal solution value also increase.

In general, a lexisearch algorithm first finds an optimal solution and then proves the optimality of that solution. The lexisearch algorithm spends a relatively large amount of time on finding optimal/best solution [23-27]. For our problem also we have the same observation. Except for the reported instances, our algorithm could not prove the optimality of the solutions within three hours of computational time. However, the results are supposed to be basis for the future study.

The algorithm certainly depends upon the structure of the instances. So a closer look at the structure of the instances and then developing a data-guided module may reduce the computational time and obtain optimal solution for the larger problem instances. Also, it is seen that the best/optimal solution is obtained very quickly, which suggests that applying a tight lower bound method may reduce the computational time drastically, which are under our investigations.

Acknowledgements

The author is very much thankful to Prof. Emdad Khan, Maharishi University of Management (MUM), USA, for his valuable comments and constructive suggestions which helped the author to improve the paper. This research was supported by Deanery of Academic Research, Al Imam Mohammad Ibn Saud Islamic University, Saudi Arabia vide Grant No. 340904.

References:

[1] VT. Paschos, An overview on polynomial approximation of np-hard problems, *Yugoslav Journal of Operations Research*, 19(1), 2009, pp. 3-40.

[2] G. Laporte, The vehicle routing problem: An overview of exact and approximate algorithms, *European Journal of Operational Research*, 59(3), 1992, pp. 345–358.

[3] GB. Dantzig, and JH. Ramser, The truck dispatching problem, *Management Science*, 6, 1959, pp. 80–91.

[4] JK. Lenstra, and AHG. Rinnooy Kan, Some simple applications of the travelling salesman problem, *Operational Research Quarterly*, 26(4), 1975, pp. 717-733.

[5] GL. Nemhauser, and LA. Wolsey, *Discrete Mathematics and Optimization*, Wiley, New York, Chichester, 1988.

[6] P. Toth, and D. Vigo, The Vehicle Routing Problem, *SIAM Monographs on Discrete Mathematics and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2002.

[7] R. Baldacci, A. Mingozzi, and R. Roberti, Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints (invited review), *European Journal of Operational Research*, 218(1), 2012, pp. 1–6.

[8] R. Baldacci, and A. Mingozzi, Lower bounds and an exact method for the Capacitated Vehicle Routing Problem, *Service Systems and Service Management*, 2, 2006, pp. 1536–1540.

[9] ML. Fisher, Optimal solution of Vehicle Routing Problems using minimum k-trees, *Operations Research*, 42, 1988, pp. 626–642.

[10] R. Baldacci, P. Toth, and D. Vigo, Recent advances in vehicle routing exact algorithms, *4OR*, 5(4), 2007, pp. 269–298.

[11] A. Pessoa, M. Poggi de Arago, and E. Uchoa, Robust branch-cut-and-price algorithms for vehicle routing problems, In: B. Golden, et al. (Eds.), *The Vehicle Routing Problem Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces Series, vol. 43, Part II, 2008, pp. 297–325.

[12] J. Gondzio, P. Gonzlez-Brevis, and P. Munari, New developments in the primal dual column generation technique, *European Journal of Operational Research*, 224(1), 2013, pp. 41–51.

[13] G. Laporte, and M. Desrochers, Two exact algorithms for the distance-constrained vehicle

routing problem, *Networks*, 14, 1984, pp. 161-172.

[14] G. Laporte, Y. Nobert, and M. Desrochers, Optimal routing under capacity and distance restrictions, *Operations Research*, 33(5), 1985, pp. 1050-1073.

[15] G. Laporte, Y. Nobert, and S. Taillefer, A branch and bound algorithm for the asymmetrical distance-constrained vehicle routing problem, *Mathematical Modelling*, 9(12), 1987, pp. 875-868.

[16] G. Clarke, and JW. Wright, Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research*, 12(4), 1964, pp. 568-581.

[17] CL. Li, D. Simchi-Levi, and M. Desrochers, On the distance constrained vehicle routing problem, *Operations Research*, 40(4), 1992, pp. 790-799.

[18] S. Almoustafa, S. Hanafi, and N. Mladenovic, New exact method for large asymmetric distance-constrained vehicle routing problem, *European Journal of Operational Research*, 226, 2013, pp. 386–394.

[19] J. Brimberg, P. Hansen, and N. Mladenovic, Attraction probabilities in variable neighborhood search, *4OR*, 8, 2010, pp. 181–194.

[20] P. Hansen, N. Mladenovic, and JA. Moreno Prez, Variable neighbourhood search: methods and applications, *Annals of Operations Research*, 175(1), 2010, pp. 367–407.

[21] N. Mladenovic, D. Urosevic, S. Hanafi, and A. Ilic, A General variable neighborhood search for the One-commodity pickup-and-delivery travelling salesman problem, *European Journal of Operational Research*, 220(1), 2012, pp. 270–285.

[22] GB. Dantzig, DR. Fulkerson, and SM. Johnson, Solution of a large-scale traveling salesman problem, *Operations Research*, 2, 1954, pp. 393–410.

[23] ZH. Ahmed, A lexisearch algorithm for the bottleneck traveling salesman problem, *International Journal of Computer Science and Security*, 3(6), 2010, pp. 569-577.

[24] ZH. Ahmed, A data-guided lexisearch algorithm for the asymmetric traveling salesman problem, *Mathematical Problems in Engineering*, Vol. 2011, Article ID 750968, 18 pages, doi:10.1155/2011/750968.

[25] ZH. Ahmed, A data-guided lexisearch algorithm for the bottleneck traveling salesman problem, *International Journal of Operational Research*, 12(1), 2011, pp. 20-33.

[26] ZH. Ahmed, An exact algorithm for the clustered traveling salesman problem, *OPSEARCH*, 50 (2), 2013, pp. 215-228.

[27] ZH Ahmed, A new reformulation and an exact algorithm for the quadratic assignment problem, *Indian Journal of Science and Technology*, 6(4), 2013, pp. 4368-4377**.**

[28] TSPLIB Website, *http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/*

Appendix

Table **4**: The search table

$\alpha_1 \rightarrow \alpha_2$	$\alpha_2 \rightarrow \alpha_3$	$\alpha_3 \rightarrow \alpha_4$	$\alpha_4 \rightarrow \alpha_5$	$\alpha_5 \rightarrow \alpha_6$	$\alpha_6 \rightarrow \alpha_7$	$\alpha_7 \rightarrow 1$
1 -> 2 (2) (0) + 18, GS	2 -> 3 (1) (2) + 17, GS	3 -> 7 (5) (3) + 12, GS	7 -> 6 (7) (8) + 7, GS	6 -> 4 (2) (15) + 12, GS 6 -> 5 (11) (15) + 15, JO	4 -> 5 (1) (17) + 11, GS	5 -> 1 (11) BS=29, Max=21, JO
			7 -> 5 (8) (8) + 13, JO			
		3 -> 5 (8) (3) + 20, JO				
	2 -> 6 (4) (2) + 20, GS	6 -> 4 (2) (6) + 23, JO				
	2 -> 7 (6) (2) + 17, GS	7 -> 6 (7) (8) + 12, GS 7 -> 5 (8) (8) + 18, JO	6 -> 4 (2) (15) + 15, JO			
	2 -> 4 (8) (2) + 20, JO					
1 -> 6 (7) (0) + 15, GS	6 -> 4 (2) (7) + 18, GS	4 -> 5 (1) (9) + 17, GS 4 -> 2 (9) (9) + 23, JO	5 -> 3 (9) (10) + 13, JO			
	6 -> 3 (5) (7) + 18, JO					
1 -> 5 (8) (0) + 21, JO STOP						

Table **5**: Results by LSA for asymmetric TSPLIB instances

Instance	**n**	**m**	$\mathbf{D_{max} = \infty}$				$\mathbf{D_{max} = 0.9*Max(1)}$			
			Max(1)	**Total**	**Ftime**	**CTime**	**Max(2)**	**Total**	**Ftime**	**CTime**
br17 (39)	17	1	39	39	4.32	37.82				
		2	39	39	2.50	65.10	31	42	13.99	160.82
Ftv33 (1286)	34	1	187	1286	17.7	53.53				
		2	1195	1302	36.40	103.74	934	1325	107.51	353.72
		3	1195	1328	30.92	620.21	934	1362	265.16	4250.68
Ftv35 (1473)	36	1	225	1473	24.94	599.18				
		2	1382	1489	29.02	1220.43	829	1491	410.08	1432.98
		3	1393	1511	908.30	7046.78	1147	1511	1013.48	7264.91
Ftv38 (1530)	39	1	209	1530	117.60	1127.35				
		2	1439	1546	320.37	2198.90	829	1548	7022.31	9976.10
		3	1467	1569	469.87	8864.32	1021	1576	5773.08	9193.51
Ftv44 (1613)	45	1	221	1613	3308.86	4928.17				
		2	1589	1615	398.61	1801.88	----	------	-------	---------
		3	1536	1654	1855.49	9041.08	1282	1654	7874.39	------

Table **6**: Results by LSA for symmetric TSPLIB instances

Instance	**n**	**m**	**$D_{max} = \infty$**				**D_{max} = 0.9*Max(1)**			
			Max(1)	**Total**	**Ftime**	**CTime**	**Max(2)**	**Total**	**Ftime**	**CTime**
burma14 (3323)	14	1	39	39	4.32	37.82				
		2	39	39	2.50	65.10	31	42	13.99	160.82
ulysses16 (6859)	16	1	1530	6840	0.05	0.18				
		2	6840	6960	0.28	0.86	5579	7011	0.23	0.78
gr17 (2085)	17	1	516	2085	0.08	0.19				
		2	2000	2188	0.00	0.53	1460	2224	0.20	0.93
gr21 (2707)	21	1	411	2707	0.00	0.01				
		2	2703	2839	0.02	0.07	2464	2953	.06	0.30
ulysses22 (7013)	22	1	1927	7013	18.23	44.24				
		2	6987	7107	13.82	117.37	5598	7165	281.15	414.39
gr24 (1272)	24	1	247	1272	5.00	7.47				
		2	1281	1389	47.41	73.37	--------	------	-------	------
fri26 (937)	26	1	181	937	5.17	14.95				
		2	904	1070	25.72	60.21	664	1085	13.94	56.72
		3	792	1214	232.45	450.21	664	1221	110.93	330.63
bayg29 (1610)	29	1	229	1610	23.22	30.18				
		2	1584	1652	5.04	28.86	1074	1678	202.21	269.06
		3	1534	1718	17.15	307.08	1074	1720	14.69	303.32
bays29 (2020)	29	1	361	2020	0.05	36.99				
		2	1351	2074	86.32	180.63	----	------	-------	---------
		3	1351	2139	53.09	675.47	1208	2279	2306.37	8525.76