Discrete Optimization

New exact method for large asymmetric distance-constrained vehicle routing problem

Samira Almoustafa\textsuperscript{a}, Said Hana\textsuperscript{b}, Nenad Mladenovi\textsuperscript{c,a,}\*

\textsuperscript{a}Brunel University, Uxbridge, UB8 3PH, UK
\textsuperscript{b}Lamih-Université de Valenciennes, France

\begin{abstract}
In this paper we revise and modify an old branch-and-bound method for solving the asymmetric distance-constrained vehicle routing problem suggested by Laporte et al. in 1987. Our modification is based on reformulating distance-constrained vehicle routing problem into a travelling salesman problem, and on using assignment problem as a lower bounding procedure. In addition, our algorithm uses the best-first strategy and new tolerance based branching rules. Since our method is fast but memory consuming, it could stop before optimality is proven. Therefore, we introduce the randomness, in case of ties, in choosing the node of the search tree. If an optimal solution is not found, we restart our procedure. As far as we know, the instances that we have solved exactly (up to 1000 customers) are much larger than the instances considered for other vehicle routing problem models from the recent literature. So, despite of its simplicity, this proposed algorithm is capable of solving the largest instances ever solved in the literature. Moreover, this approach is general and may be used for solving other types of vehicle routing problems.
\end{abstract}

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

The vehicle routing problem (VRP) is defined as follows: 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. It is an NP-hard problem (Haimovich et al., 1988; Paschos, 2009). There are many kinds of VRPs. For an overview of VRPs variants and applications, we refer to Nemhauser and Wolsey (1988), Laporte (1992), Toth and Vigo (2002), Baldacci et al. (2007), Laporte (2007), Baldacci et al. (2012). We consider in this paper 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 node \(i\) to node \(j\) differs from that of node \(j\) to node \(i\), we call this problem asymmetric (ADVRP), otherwise it is defined as symmetric DVRP. Surprisingly ADVRP, asymmetric problems have not been studied as much as the other types of VRPs. To the best of our knowledge, the last such paper was published by Laporte et al. (1987).

Variety of integer programming formulations have been proposed for VRPs, including the so-called two- and three-index formulations, the set partitioning formulation, and those formulations based on extra variables representing the flow of one or more commodities (see e.g., survey and formulation comparisons in Letchford and Salazar-González (2006)). On the other hand, recent solution techniques are mostly based on branch-and-cut, branch-and-cut-and-price, or column generation (see Baldacci et al. (2007); Pessoa et al. (2008); Gondzio et al. (2013) and recent survey by Baldacci et al. (2012)).

Three types of algorithms are used to solve any VRP. The first type consists of exact algorithms that are time-consuming, which has been studied by Lenstra and Rinnooy Kan (1975), Christofides et al. (1981), Laporte et al. (1985), Laporte and Nobert (1987), Baldacci et al. (2012), etc. The second type consists of classical heuristics such as greedy, local search, and relaxation based, also studied by many researchers, e.g., (Clarke and Wright, 1964; Lawer et al., 1985; Toth and Vigo, 2002). These heuristics produce approximate solution faster, when compared with the first type, but without guarantee of optimality. The third type consists of heuristics that are based on some metaheuristic rules. Such meta-heuristics or framework for building heuristics are Simulated annealing, Tabu search, Genetic algorithms (Toth and Vigo, 2002), Variable neighborhood search (Brimberg et al., 2010; Hansen et al., 2010; Mladenovic et al., 2012a), etc.

In this paper, we suggest a new simple algorithm for solving ADVRP based on Branch and Bound (B&B) method. As in Laporte et al. (1987), the ADVRP is first transformed into the Travelling salesman problem (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 node.
of the search B&B tree. We use the best-first-search strategy and adapted tolerance based rules for branching. This means that the next node in the tree is the one with the smallest relaxed objective function value. In the case of tie, we use two tie-breaking rules: (i) take the last one from the list; (ii) take one from the list at random. Since the random tie-breaking rule gives better results, we do not consider the deterministic rule in this paper. We found that our B&B based method is very fast but memory consuming. That is why, we suggested in this paper multistart B&B method (MSBB-ADVRP). It simply uses random tie-breaking rule in selection of the next subproblem. Computational results show that we are able to provide exact solutions for instances with up to 1000 nodes. The size of problems could be even larger if more powerful computer (with larger memory) is used. As far as we know, these instances are much larger than the instances considered in the recent literature for other similar VRP models and exact solution approaches. For example, in Baldacci and Mingozzi (2009), several VRP problem types are studied and sophisticated exact solution methods tested. The largest instances solved had 199 customers. Therefore, our simple algorithms are capable of solving the largest instances ever solved in the literature.

The structure of this paper is as follows. In Section 2, we present mathematical programming formulations of ADVRP. In Section 3, we explain details of single and multistart B&B for solving ADVRP. In Section 4, we present illustrative example for solving ADVRP by single B&B. Section 5 contains computational results. In Section 6, we give conclusions and future research directions.

2. Mathematical programming formulations for ADVRP

In this section, we give two mathematical programming formulations of ADVRP. The first, so-called flow based formulation is used for comparison purposes in the computational results section. The second is based on transformation of ADVRP to asymmetric travelling salesman problem (TSP). We use its relaxation in our B&B exact method, which will be described in Section 3.

Let \( N = \{1, 2, \ldots, n - 1\} \) denote the set of customers and \( V = N \cup \{0\} \) the set of nodes, where 0 is the index of the depot. A set of arcs is denoted by \( A' \), \( A' = \{(i, j) \in V \times V : i \neq j\} \). The matrix \( D' \) is defined as \( D' = [d'_{ij}] \) where \( d'_{ij} \) presents the travelled distance from node \( i \) to node \( j \). The number of vehicles and the maximum distance allowed are denoted by \( m \) and \( D_{\text{max}} \), respectively. The shortest distance between node \( i \) and node \( j \) is denoted by \( c_{ij} \). The decision binary variable \( x_{ij} \) is defined as follows:

\[
x_{ij} = \begin{cases} 
1 & \text{if the arc } (i, j) \text{ belongs to any tour and } i \neq j; \\
0 & \text{otherwise}. 
\end{cases}
\]

2.1. Flow based formulation

For the sake of comparison, we use another formulation of ADVRP with polynomial number of variables and constraints, without copying the depots. This is achieved by introducing a new set of variables \( z_{ij} \) that presents the shortest length travelled from the depot to customer \( j \), where \( i \) is the predecessor of \( j \). The formulation of ADVRP, which will be later used with CPLEX solver (CPLEX-ADVRP), is given below (Kara et al., 2007; Kara, 2011):

\[
f(S) = \min \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} x_{ij}
\]

subject to

\[
\sum_{i=0}^{n-1} x_{ij} = 1 \quad j = 1, 2, \ldots, n - 1
\]

\[
\sum_{j=0}^{n-1} x_{ij} = 1 \quad i = 1, 2, \ldots, n - 1
\]

\[
\sum_{i=0}^{n-1} x_{ij} = m
\]

\[
\sum_{j=0}^{n-1} x_{ij} = m
\]

\[
x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A
\]

Obviously, there is a polynomial number of variables and constraints. This model is known as a flow based model since the constraint (7) is a typical flow constraint. It says that the distance from depot to \( i \) any other node \( j \) on the tour should be equal to the difference between the distance from depot to \( i \) and the distance from depot to \( j \). The constraint (8) demands that the travelled distance from depot to node \( j \) is less than or equal to the difference between the maximum distance allowed and the distance from node \( j \) to depot. The constraint (9) checks that the total distance travelled up to the depot is less than or equal to the maximum distance allowed. In addition, according to the constraint (10), the total distance from depot to node \( j \) should be greater than or equal to the distance from the depot to node \( i \) plus the distance from node \( i \) to node \( j \). The constraint (11) gives the initial value for \( z_{0i} \), which is equal to the distance from the depot to node \( i \). The last constraint (12) introduces the decision variables \( x_{ij} \) as binary variables.

2.2. TSP formulation

The TSP formulation may be obtained by adding \( m - 1 \) copies of the depot to \( V \) (Lenstra and Rinnooy Kan, 1975). Now, there are \( n + m - 1 \) nodes in the new augmented directed graph \( G(V, A) \), where \( V = V \cup \{n + 1, \ldots, n + m - 2\} \).

The distance matrix \( D = [d_{ij}] \) is obtained from \( D' \) by the following transformation rules where \( i, j \in V \):

\[
d'_{ij} = \begin{cases} 
1 & \text{if } 0 \leq i < n, 0 \leq j < n, i \neq j;

d'_{ij} & \text{if } i \geq 0, n \leq j < n;

d'_{ij} & \text{if } 0 < i < n, j \geq n;
\infty & \text{otherwise}
\end{cases}
\]

Then the formulation of TSP (Dantzig et al., 1954) given below (13)–(16) is as follows:

\[
f(S) = \min \sum_{(i, j) \in A} d_{ij} x_{ij}
\]

where \( x_{ij} \) satisfies these conditions

\[
\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V
\]

\[
\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V
\]

\[
\sum_{i \in V \setminus \{0\}} x_{ij} \leq |U| - 1 \quad \forall U \subset V \setminus \{0\}, \ |U| \geq 2.
\]

distance constraints
The constraints (14) and (15) ensure that in and out degree of each node are equal to 1. The constraint (16) eliminates subtours, where \( U \) is any subset of \( V \). To formulate ADVRP in addition to (13)–(16), we need to add distance constraint (17) to check if the total distance for each tour is less than maximum distance allowed (\( D_{\text{max}} \)).

The weak point of this formulation is the exponential number of constraints in (16), since the number of subsets \( U \) is exponential. However, in our B&B method, this will be explained in the next section, this set of constraints will be relaxed.

DVRP may also be seen as a special case of VRP with time windows constraints (Letchford and Salazar-González, 2006). In the vehicle routing problem with time windows (VRPTW), it takes some time \( t_{ij} \geq 0 \) to traverse arc \((i,j)\). When \( i \in N \), the value \( t_{ij} \) includes any time required to service customer \( i \). For each customer \( i \in N \), the service must begin within the time window \([e_i, l_i] \), where \( 0 \leq e_i \leq l_i \leq \infty \). We will also allow each vehicle to leave the depot at time \( t_0 \) or afterwards, and to come back to the depot on time \( t_f \) or earlier. A vehicle is permitted to wait at a node, either before or after serving a customer. The DVRP can be viewed as a special case of the VRPTW by setting \( t_{ij} = d_{ij} \), \( e_i = d_{i0} \) and \( l_i = D - d_{i0} \).

3. New exact method for ADVRP

Here, we first give steps of branch and bound (B&B) with tolerance based branching rule and random selection of the next subproblem in case of ties.

3.1. Single start B&B for ADVRP (RND-ADVRP)

The branch and bound (B&B) is an exact method for solving integer programming problem. It consists of enumerating all possible solutions within the so-called search tree, and pruning subtrees when better solutions than the current one (upper bound) cannot be found. B&B rules for solving minimization problem are briefly given below:

- The original problem is placed at the root of the search tree. All other nodes represent the subproblems. In solving the subproblems, to obtain the lower bound, some variables are fixed and some constraints are ignored (relaxed).
- Each node in the search tree contains lower (LB) and upper bound (UB) values and the corresponding solutions. LB solution is not necessarily feasible. It is obtained by neglecting (relaxing) some constraints. UB solution is feasible. During iterations, the gap between LB and UB decreases;
- UB presents the value of the best feasible solution found so far. To get the initial value of UB, we can use some heuristic. If the heuristic solution is not available, then the upper bound is set to infinity (UB = \( \infty \)). The value of UB will be updated during the search whenever feasible solution with smaller value is found;
- The search strategy defines the way in which we choose the next node for branching. There are three basic branching strategies: breadth first search, depth first search and hybrid search, which is also called best first search strategy. In this article, we implement this strategy.

3.1.1. Lower bounds

To apply B&B for ADVRP, we need a lower bounding procedure that should be performed at each node of the search tree. Of course, there are many ways to relax model (13)–(17). The more constraints are included, the better (higher) lower bounds are got. Really, the set of feasible solutions of the problem with \( m \) constraints is a subset of feasible solutions of the same problem with \( m - 1 \) constraints. Thus, its objective function value is worse. However, adding new constraints makes the lower bound higher. We use Assignment problem (AP) as a lower bounding procedure of the TSP formulation given in (13)–(15), i.e., we relax all tour elimination and maximum distance constraints (16) and (17). Although the quality of the lower bound is not high, the benefit is in using very fast exact Hungarian method (Kuhn, 1955) for solving AP. Here, we use the implementation described in Jonker and Volgenant (1986). The complexity of AP at root node is in \( O(n^2) \) (Volgenant, 2006). Another advantage is that the relaxed AP solution is already an integer.

**Proposition 1.** Any feasible AP solution of the problem (13)–(15) consists of a set of cycles, i.e., a sequence of arcs starting and ending at the same vertex with the number of arcs in any cycle \( k \) is greater than or equal to 2 (\( \omega_k \geq 2 \)).

**Proof.** It is clear from (14) and (15) that the degree of each vertex in \( S \) is equal to 2. It has one in-coming and one out-going arc. If the matrix \( D \) was symmetric and \( n \) even, then those cycles would contain just 2 vertices and therefore, \( \omega_k = 2 \). However, in all other cases, \( \omega_k \) is obviously larger than 2: if vertex \( i \) is assigned to \( j \), then \( j \) is not necessarily assigned to \( i \). □

There are three types of cycles obtained by AP relaxation:

- a served cycle – contains exactly one depot;
- an unserved cycle – contains no depot;
- a path – contains more than one depot.

In the last case, each path may be divided into served cycles. Therefore, the number of served cycles is equal to the number of depots in the path. Subsequently, the term tour is used to denote either a served cycle, an unserved cycle, or a path; the term depot is used to denote either the original depot or a copy of the depot. A tour is called infeasible if its total distance is larger than \( D_{\text{max}} \), or if it contains no depot.

3.1.2. Branching rule

Since the sets of constraints (16) and (17) are relaxed, the AP solution may have many infeasible tours. If the tour is infeasible, it must be destroyed, i.e., one arc should be excluded (deleted). We exclude an arc from the current infeasible solution \( S \) by giving it a large value (\( \infty \)), and then resolving AP relaxation again. Really, in the new solution, such an arc will not appear since we minimize AP objective function. There are several ways to remove an arc from \( S \). We can try all possible removals (one at the time) and collect all objective function values obtained from solving new corresponding APs.

However, in this paper, we use the concept of tolerance. Tolerance is one of the sensitivity analysis techniques, for more details on sensitivity analysis see (Kolotai and Terlaky, 2000; Lin and Wen, 2003). The definition of tolerance is used as a branching rule within B&B method in Turkensteen et al. (2008) for solving ATSP. We first extend this idea for solving ADVRP.

The difference between the value of the objective function before and after the exclusion of an arc in the current solution is called upper tolerance (UT) of the arc (Goldengorin et al., 2006). The arc to be removed corresponds to the smallest objective function value obtained. Therefore, the arc which has the smallest upper tolerance is chosen to be excluded in our branching rule. Some preliminary results of this approach have been given in Almoustafa et al. (2009).

3.1.3. Notation

In order to explain the steps of RND-ADVRP algorithm, we use the following notation:
Algorithm 1. (RND-ADVRP) Algorithm

Given an instance of ADVRP \( (n,m,D_{\text{max}},S^*,\text{ind}) \):  
1. Initialize UB = \( m \times D_{\text{max}} \), APcnt = 1, set iteration counter \( i = 1 \);  
2. Solve AP problem using HA to obtain a solution \( S = \{T_k\}_{k=1}^n \);  
3. \( L = \{1\} \) – the list contains the root node;  
4. Calculate \( d(T_k) \) and \( t(T_k) \) for every tour \( T_k \in S \);  
5. If \( S \) feasible then \( S' = S \) is an optimal solution; \( \text{ind} = 1 \); stop;  
   while (APcnt < Maxnodes) do  
6. \text{Branching.} Choose a subproblem \( b_i \in L \) with the smallest value of the objective function \( b_i = \min_{b_i \in L} \{f(b_i)\} \); in the case of tie, choose one from the list at random;  
7. \text{Best first.} Find the ratio \( d(T_k)/t(T_k) \) for every infeasible tour \( T_k \in S' \);  
8. \text{Tolerance (expanding search tree).} Calculate upper tolerances for all arcs in this \( T_k \) tour by solving \( t(T_k) \) times AP problem to get solutions \( S_r \) where \( r = 1, \ldots, t(T_k) \). Expand search tree with those \( t(T_k) \) subproblems, and update APcnt as:  
   \[ \text{APcnt} = \text{APcnt} + t(T_k) \];  
9. \text{Feasibility check.} Remove \( b_i \) from \( L \). Check feasibility of all new expanded nodes. If feasible, update UB if necessary;  
10. \text{Update.} If UB is updated, then update \( L \) based on the new UB value as:  
   \[ L = L \cup \{f(S_r) < \text{UB}\} \setminus \{f(S_r) > \text{UB}\} \] where \( S_r \) are the new generated infeasible solutions, and \( S_q \) are the existing infeasible solutions. Otherwise, update \( L \) based on the current UB value as:  
   \[ L = L \cup \{f(S_q) < \text{UB}\} \];  
11. \text{Optimality conditions.} If \( L = \emptyset \) and UB = \( m \times D_{\text{max}} \) then \( S' \) is the optimal solution where \( f(S') = \text{UB} \);  

3.1.4. Algorithm RND-ADVRP

Here, we briefly explain the steps of the algorithm RND-ADVRP (see Algorithm 1). At the root node, we find solution \( S \) by solving AP problem. Then, we calculate the total distance \( d(T_k) \) for every tour \( T_k \) of \( S \) and check the feasibility of the solution. If it is feasible, then the optimal solution is found, and the program stops. Otherwise, we repeat the following steps until the memory limit is reached:

- **Branching.** Choose the subproblem \( b_i \in L \) with the smallest value of the objective function, where \( i \) denotes the iteration number. In the case when more than one subproblem has the same smallest value, choose one from the list of active subproblems \( L \) at random.
- **Best first.** Find the ratio between the total distance \( d(T_k) \) and the number of arcs in the chosen subproblem \( t(T_k) \) for every infeasible tour \( T_k \), \( k = 1, \ldots, M \) where \( M \) is the number of infeasible tours. Choose the tour \( T_k \) with the largest ratio \( d(T_k)/t(T_k) \).
- **Tolerance (expanding search tree).** Calculate upper tolerances for all arcs in this tour \( T_k \) as follows. Exclude in turn one arc from \( T_k \) by putting the value of \( \infty \) in the distance matrix at the corresponding position. Find AP solution to these subproblems. For each excluded arc find the difference between the value of the objective function before and after excluding the arc. This gives the upper tolerance (UT) value to that arc. Note that the value of the counter APcnt is increased by the number of arcs in the chosen tour:  
   \[ \text{APcnt} = \text{APcnt} + t(T_k) \]
- **Check Feasibility.** Rem violence the set of all new generated infeasible solutions, and \( S_q \) are the existing infeasible solutions. Otherwise, i.e., if the upper bound is not updated, then update UB by adding the new expanded subproblems that have value smaller than UB and by removing those subproblems that have value greater than UB as follows:  
   \[ L = L \cup \{f(S_r) < \text{UB}\} \setminus \{f(S_q) > \text{UB}\} \]
   Note that \( S_r \) are the new generated infeasible solutions, and \( S_q \) are the existing infeasible solutions. Otherwise, i.e., if the upper bound is not updated, then update UB by adding the new expanded subproblems that have value smaller than UB as follows:

   \[ L = L \cup \{f(S_r) < \text{UB}\} \]
- **Optimality conditions.** Check if \( L = \emptyset \) and UB is updated, then stop with the value of optimal solution \( f(S') = \text{UB} \); Stop otherwise, if \( L = \emptyset \) and UB is not updated, stop with the message that no feasible solution exists (\( \text{ind} = 4 \)).
- **Termination.** When there is no memory, we get two possible outputs: If UB is updated, then \( S' \) is returned as a feasible but not proved as the optimal solution (\( \text{ind} = 2 \)). Otherwise, a feasible solution has not been found, (but it might exists (\( \text{ind} = 3 \)).
Proposition 2. If there is no memory limit, then the algorithm RND-ADVRP finds an optimal solution to ADVRP or proves that such a solution does not exist.

Proof. Let us denote with $F$ and $G$ the set of all feasible solutions of ADVRP problem and the set of solutions generated by our B&B algorithm, respectively. In order to show that RND-ADVRP works properly, it is enough to show that $F \subseteq G$. In other words, we need to show that no feasible ADVRP solution is omitted in enumeration of cycles produced by AP relaxation. Really, our enumeration is based on arcs elimination (by giving them value $\infty$ in step 8) and decreasing the number of tours. It is followed by solving lower bounding AP problem. AP provides a solution $S$ as a set of cycles (Proposition 1). Clearly, the set of all solutions $G$ generated by our method contains all possible cycles with $m$ vehicles, and therefore all tours from $F$. Thus, the set of all feasible ADVRP tours is the subset of all this generated sets, and our RND-ADVRP enumerates all feasible tours. □

3.2. Multistart B&B for ADVRP (MSBB-ADVRP)

The idea of (RND-ADVRP) is to introduce random selection of the next subproblem among those with the same (smallest) objective function value. This random choice may cause the generation of smaller search tree. Therefore, if we reach the maximum number of subproblems allowed, we restart the exact B&B method hoping that in the next attempt, we will get an optimal solution. For that reason, we rerun RND-ADVRP many times until an optimal solution is found or infeasibility is proven. MSBB-ADVRP algorithm is summarized as follows (see Algorithm 2):

Algorithm 2. (MSBB-ADVRP) Algorithm

<table>
<thead>
<tr>
<th>Algorithm MSBB-ADVRP($n,m,D_{\text{max}},D,\text{Maxnodes},\text{ntrail},S_{\text{best}}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $f_{\text{best}} \leftarrow \infty$;</td>
</tr>
<tr>
<td>2 for $i = 1$ to ntrail do</td>
</tr>
<tr>
<td>3 RND-ADVRP($n,m,D_{\text{max}},D,\text{Maxnodes},S',\text{ind}$);</td>
</tr>
<tr>
<td>4 if ($\text{ind} = 1$) then ($S_{\text{best}} = S'$; stop);</td>
</tr>
<tr>
<td>5 if ($\text{ind} = 4$) then (stop);</td>
</tr>
<tr>
<td>6 if $1 &lt; \text{ind} &lt; 4$ then</td>
</tr>
<tr>
<td>7 if ($f(S') &lt; f_{\text{best}}$) then</td>
</tr>
<tr>
<td>8 $S_{\text{best}} = S'; f_{\text{best}} = f(S')$;</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Each instance is run a given number of times ($\text{ntrail}$ – a parameter); stop when an optimal solution is found ($\text{ind} = 1$), or infeasibility of the problem instance is proven ($\text{ind} = 4$). This will increase the chances to find an optimal solution, or at least to improve the value of the best feasible solution found so far. The number of reruns needs to be given by the user.

4. Illustrative example

The example, given in Table 1, is taken from Balas and Toth (1985), where the number of customers is $n = 8$ and the number of vehicles is $m = 2$. In matrix $D$, the first row represents the distances from the depot to all other customers. The first column represents the distances from each customer to the depot, and all other entries represent distances between the customers. To solve this problem, we have to add $m = 2 = 2 - 1 = 1$ copy of the depot. Table 2 illustrates the new distance matrix after adding the last row and the last column according to the new distance function (see Section 2: “TSP formulation”). In the new matrix 0 and 8 are indices of the depot. We will consider 2 problems with this dataset, i.e., we will have 2 different values of the parameter $D_{\text{max}}$.

Lower bound (LB) solution at the root node of B&B tree, obtained by solving AP (13)–(15), is given in Fig. 1a. Each depot label is written inside a squared box, and the total distance is written inside each tour.

Problem 1. First, the value of $D_{\text{max}}$ is set to $\infty$. The AP solution counter, the list of active subproblems in the search tree, and the iteration counter are initialized ($AP\text{cnt} = 1, L = \{1\}, i = 1$). Clearly, the initial value of $UB = m \times D_{\text{max}} = \infty$. Since $b_1 \in L$, we have $b_1 = 1$.

Iteration 1. It can be seen from Fig. 1a that the AP solution $S_1$ has three tours: $T_1 = \{(0,6),(6,7),(7,0)\}; T_2 = \{(8,1),(1,2),(2,8)\}; and$ finally $T_3 = \{(5,3),(3,4),(4,5)\}$. One of them is infeasible, since it does not contain the depot (node 0 or 8), thus solution $S_1$ is not feasible.

The program will check the total distance for all tours in the solution at the root node of the search tree: $d(T_1) = (d(T_1), d(T_2), d(T_3)) = (16,8,5)$. The corresponding objective function value is $f_1 = f(S_1) = 16 + 8 + 5 = 29$. Since only the tour $T_2$ is infeasible, we choose arcs from $T_3$ for branching. The number of arcs in $T_3$ is equal to 3 ($\#(T_3) = 3$) and its total distance is equal to 5 ($d(T_3) = 5$). The value of the upper tolerance for each arc of $T_3 = \{(5,3),(3,4),(4,5)\}$ should be calculated in the following way:

(i) Arc (5,3): exclude this arc from the solution, i.e., replace its value $d(5,3) = 2$ with $d(5,3) = \infty$, and then solve the AP. The resulting solution is given in Fig. 1b. This new solution is not feasible as well, since there are two infeasible tours $T_1 = \{(3,4),(4,3)\}; and$ $T_2 = \{(6,7),(7,6)\}$. The value of the optimal AP solution at node 2 in the search tree is $f_2 = f(S_2) = 17 + 8 + 5 + 4 = 34$. The UF value for the arc (3,4) in the solution $S_2$ is calculated as: $f_2 - f_1 = 34 - 29 = 5$.

(ii) Arc (3,4): first, restore the value of $d(3,4)$ into its previous value 2. By excluding an arc (3,4) as before, we get $f_3 = f(S_3) = 8 + 23 + 4 = 35$. The solution at node 3 is also not ADVRP feasible since it contains one infeasible tour $T_3$ (Fig. 1c). The value of the upper tolerance is $f_3 - f_2 = 35 - 29 = 6$.

(iii) Arc (4,5): as before, restore $d(3,4)$ to its previous value 1. Excluding the arc (4,5) produces $f_4 = f(S_4) = 21 + 8 + 4 = 33$ (Fig. 1d). The solution at node 4 is not ADVRP feasible since it contains one infeasible tour $T_4$. The value of upper tolerance for the arc (4,5) is: $f_4 - f_3 = 33 - 29 = 4$.

Table 1

<table>
<thead>
<tr>
<th>Original distance matrix for ADVRP with $n = 8$ and $m = 2$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>New distance matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
The search tree node counter is increased by 3: $\text{APcnt} = 1 + 3 = 4$.

The upper bound (UB) is not updated since no feasible solution is found. Thus, the new list of active subproblems is $L = \{2, 3, 4\}$. Since the condition $f(S_r) < \infty$, $(r = 2, 3, 4)$ holds for all 3 subproblems above.

**Iteration 2.** The smallest value among three values obtained by $f(S_2) = 34$, $f(S_3) = 35$, $f(S_4) = 33$ in iteration 1 was 33 and it corresponds to exclusion of the arc $(4,5)$. Thus, the index of the next subproblem $b_2 = 4$, see Fig. 2a and b. The corresponding solution is infeasible (see Fig. 1d). Among the three tours in that solution $S$, only one is infeasible. It has two arcs $T_3 = \{(7,6),(6,7)\}$. Thus, in this case, the two new subproblems are generated:

(i) **Arc (7,6):** exclude this arc $(d(T_3) = \infty)$ and solve the corresponding AP. Note that two arcs are now excluded: $(4,5)$ and $(7,6)$. Then we get a feasible solution $S_5 = \{T_1,T_2\}$ where:

$T_1 = \{(0,6),(6,7),(7,5),(5,3),(3,4),(4,0)\}$;

$T_2 = \{(8,1),(1,2),(2,8)\}$, and $d(T_1) = 26$, $d(T_2) = 8$, $f_5 = f(S_5) = 34$ (see Fig. 3a).

(ii) **Arc (6,7):** restore the value of arc $(7,6)$ to its previous value $(d(T_1) = 26 > D_{\text{max}} = 23)$. As can be seen in Fig. 2b, we have two nodes of the search tree (node 2 and node 5) that have the same smallest objective function value 34. Choose one of them at random to branch next. Assume that $b_3 = 5$. Since there are 6 arcs in the infeasible tour $T_1$ of $S_5$, there are 6 new subproblems generated in the search tree (see Fig. 2c). When we branch at node 5, we find the first feasible solution at node 9, so we update UB = 37, $\text{APcnt} = 12$, $L = \{2, 3, 7, 10, 12\}$.

**Iteration 3.** In this iteration we have 3 nodes (2, 7, and 12) that have the same smallest value 34 (see nodes 2, 7 and 12 in Fig. 2c). So, we need to choose one among them at random, e.g., $b_4 = 2$. This step is repeated at each iteration until the optimal solution is found. At the end, we get optimal solution value 37, $f^* = f(S^*) = 37$ (see Fig. 3b).

**Problem 2.** The longest tour in the optimal solution of Problem 1 is 26, so the new value of $D_{\text{max}}$ is obtained as $D_{\text{max}} = 0.9 \times 26 = 23.4$. We choose $D_{\text{max}} = 23$ and run the same example.

**Iteration 1.** This iteration is the same as iteration 1 in Problem 1, i.e., $L = \{2, 3, 4\}$.

**Iteration 2.** In this iteration $L = \{2, 3, 5, 6\}$ since the solution $S_5$ is no more feasible $(d(T_1) = 26 > D_{\text{max}} = 23)$. As can be seen in Fig. 2b, we have two nodes of the search tree (node 2 and node 5) that have the same smallest objective function value 34. Choose one of them at random to branch next. Assume that $b_3 = 5$. Since there are 6 arcs in the infeasible tour $T_1$ of $S_5$ (see Fig. 3a) then there are 6 new subproblems generated in the search tree (see Fig. 2c). When we branch at node 5, we find the first feasible solution at node 9, so we update UB = 37, $\text{APcnt} = 12$, $L = \{2, 3, 7, 10, 12\}$.

**Iteration 3.** In this iteration we have 3 nodes (2, 7, and 12) that have the same smallest value 34 (see nodes 2, 7 and 12 in Fig. 2c). So, we need to choose one among them at random, e.g., $b_4 = 2$. This step is repeated at each iteration until the optimal solution is found. At the end, we get optimal solution value 37, $f^* = f(S^*) = 37$ (see Fig. 3b).
5. Computational results

5.1. Computers

All experiments were implemented under Windows XP and on Intel(R) Core(TM) 2 CPU 6600@2.40 gigahertz, with 3.24 gigabytes of RAM. The code is written in C++ language. Some parts of the code are taken from Turkensteen et al. (2008).

5.2. Test instances

Full asymmetric distance matrices are generated at random using uniform distribution to generate integer numbers between 1 and 100. The generator for random test instances needs the following input data:

- \( n \) – the size of distance matrix;
- \( \gamma \) – the parameter that controls the degree of symmetry in the distance matrix, \( \gamma \in [0, 1] \): 0 means completely random and asymmetric; 1 means completely symmetric, etc.

For \( n \leq 200 \), four different instances were generated for each combination of \((n, m)\). However, only one distance matrix is generated for the cases where: \( 200 < n \leq 1000 \). In addition, the shortest distances between each two customers are calculated to get input matrix \( C \). Test instances are divided into two groups: small size \((n = 40, 60, \ldots, 200)\), and large size \((n = 240, 280, \ldots, 1000)\) instances. For each small size matrix, two different types of instances are generated based on the different number of vehicles: the case where \( m_1 = n/20 \) and the case where \( m_2 = n/10 \). For instances belonging to the large size set, we use only \( m_1 \). In addition, for each distance matrix, we consider three problems with three different values of \( D_{\text{max}} \). First value of \( D_{\text{max}} \) is \( \infty \), then we use this formulae to obtain new value of \( D_{\text{max}} \):

\[
D_{\text{max}}(i) = 0.90 \times LT(i - 1), \quad i \in \{2, 3\}, \quad \text{and} \quad LT(i - 1) \text{ is the longest tour in the optimal solution when the value of maximum distance allowed is } D_{\text{max}(i-1)}. \]

Thus, the total number of instances is 257. All test instances used in this paper can be found on the web site http://www.mi.sanu.ac.rs/~nenad/advrp/ as well as the code for generator coded in C++.

5.3. Methods compared

In this paper, we compare three methods to find the optimal solution to ADVRP: CPLEX-ADVRP, RND-ADVRP, MSBB-ADVRP. In all our experiments reported below, we run MSBB-ADVRP five times. We note that by increasing the number of restarts, chances to find the optimal solution might improve but with the cost of larger CPU time. In CPLEX-ADVRP the process will continue until the optimal solution is found, or the time limit of 3 hours (10,800 seconds) is reached.

Comparison. Tables 3–5 contain summary results to all 257 test instances from \( n = 40 \) up to \( n = 1000 \) customers with \( D_{\text{max}(1)} = \infty \), \( D_{\text{max}(2)} = 0.90 \times LT(1) \), and \( D_{\text{max}(3)} = 0.90 \times LT(2) \) respectively. The rows in all tables give the following characteristics:
1. $\text{Opt (ind } = 1\text{)}$ – how many times each program finds the optimal solution;
2. $\text{Feas (ind } = 2\text{)}$ – how many times feasible (but not optimal) solutions have been found;
3. $\text{No Mem (ind } = 3\text{)}$ – how many times feasible solutions are not found because of the lack of memory;
4. $\text{No Feas (ind } = 4\text{)}$ – how many instances with proven infeasibility are detected;
5. Total time in seconds – the total time in seconds spent only for instances where the optimal solutions are found;
6. Average time – the average time for instances for which the optimal solution is found, where (Average time = Total time/ $\text{Opt}$);

Detailed results for all three methods may be found on our web site http://www.mi.sanu.ac.rs/~nenad/advrp/.

The numerical analysis identifies:

(i) The most effective method on average is our Multistart Branch and Bound for ADVRP (MSBB-ADVRP). For 92 instances in the first stage with $D_{\text{max}}(1) = 1$, the rate of success are: 80% for CPLEX-ADVRP, 100% for RND-ADVRP and 100% for MSBB-ADVRP. For the second stage with $D_{\text{max}}(2) = 0.90 \times L(1)$ the rate of success are: 78%, 77%, and 79% for CPLEX-ADVRP, RND-ADVRP, and MSBB-ADVRP respectively. Finally, in the third stage with $D_{\text{max}}(3) = 0.90 \times L(2)$ the rate of success for the three programs CPLEX-ADVRP, RND-ADVRP, and MSBB-ADVRP are: 71%, 74%, and 77% respectively.

(ii) Regarding efficiency (cpu time), it can be seen that RND-ADVRP and MSBB-ADVRP are much faster than CPLEX-ADVRP. In addition, RND-ADVRP is slightly faster than MSBB-ADVRP. The average time RND-ADVRP spent for solving Problem 1, Problem 2, and Problem 3 are in all stages are 0.08, 2.21, and 31.99 seconds. The corresponding average time for MSBB-ADVRP are 0.09, 3.58, and 36.51 seconds, while the time for CPLEX-ADVRP are 346.79, 745.06, and 1055.81 seconds. However, in total, the number of instances solved exactly by MSBB-ADVRP are larger than the number of instances solved by RND-ADVRP and CPLEX-ADVRP. We notice that RND-ADVRP and MSBB-ADVRP produce similar results in the first two stages (problems). When the problem becomes harder ($D_{\text{max}}$ smaller), the gap between RND-ADVRP and MSBB-ADVRP increases. In solving larger instances MSBB-ADVRP takes benefit of multistart. The reason why our methods are faster than CPLEX-ADVRP is the fact that in our approach the problem specific knowledge is included, while the CPLEX is general solver: we use fast polynomial Hungarian method as lower bounding routine; we use problem specific data structure;

| Table 3 | Results for 92 instances with $D_{\text{max}}(1) = \infty$. |
| --- | --- | --- |
| 72 Small test inst. | 20 Large test inst. | Total of 92 inst. |
| CPLEX | RND | MSBB | CPLEX | RND | MSBB | CPLEX | RND | MSBB |
| $\text{Opt (ind } = 1\text{)}$ | 72 | 72 | 72 | 2 | 20 | 20 | 74 | 92 | 92 |
| $\text{Feas (ind } = 2\text{)}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\text{No Mem (ind } = 3\text{)}$ | 0 | 0 | 0 | 18 | 0 | 0 | 18 | 0 | 0 |
| $\text{No Feas (ind } = 4\text{)}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Total time in seconds | 21896.33 | 0.79 | 0.77 | 3766.31 | 6.89 | 7.52 | 25662.64 | 7.68 | 8.29 |
| Average time | 304.12 | 0.01 | 0.01 | 1883.16 | 0.34 | 0.38 | 346.79 | 0.08 | 0.09 |
| $\% \text{ Of solved}$ | 100 | 100 | 100 | 10 | 100 | 100 | 80 | 100 | 100 |

| Table 4 | Results for 92 instances with $D_{\text{max}}(2) = 0.90 \times L(1)$. |
| --- | --- | --- |
| 72 Small test inst. | 20 Large test inst. | Total of 92 inst. |
| CPLEX | RND | MSBB | CPLEX | RND | MSBB | CPLEX | RND | MSBB |
| $\text{Opt (ind } = 1\text{)}$ | 70 | 51 | 53 | 2 | 20 | 20 | 72 | 71 | 73 |
| $\text{Feas (ind } = 2\text{)}$ | 2 | 17 | 16 | 2 | 0 | 0 | 2 | 17 | 16 |
| $\text{No Mem (ind } = 3\text{)}$ | 0 | 2 | 1 | 16 | 0 | 0 | 16 | 2 | 1 |
| $\text{No Feas (ind } = 4\text{)}$ | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 2 | 2 |
| Total time in seconds | 409723 | 33 | 140 | 12672 | 124 | 122 | 53645 | 157 | 262 |
| Average time | 569.06 | 0.46 | 1.94 | 633.59 | 6.22 | 6.09 | 745.06 | 2.21 | 3.58 |
| $\% \text{ Of solved}$ | 97 | 71 | 74 | 95 | 95 | 95 | 78 | 77 | 79 |

| Table 5 | Results for 73 instances with $D_{\text{max}}(3) = 0.90 \times L(2)$. |
| --- | --- | --- |
| 53 Small test inst. | 20 Large test inst. | Total of 73 inst. |
| CPLEX | RND | MSBB | CPLEX | RND | MSBB | CPLEX | RND | MSBB |
| $\text{Opt (ind } = 1\text{)}$ | 51 | 15 | 37 | 1 | 19 | 19 | 52 | 54 | 56 |
| $\text{Feas (ind } = 2\text{)}$ | 2 | 15 | 13 | 0 | 0 | 0 | 2 | 15 | 13 |
| $\text{No Mem (ind } = 3\text{)}$ | 0 | 3 | 3 | 19 | 1 | 1 | 19 | 4 | 4 |
| $\text{No Feas (ind } = 4\text{)}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Total time in seconds | 50870 | 69 | 397 | 4032 | 1659 | 1647 | 54902 | 1727 | 2045 |
| Average time | 997 | 2 | 11 | 4032 | 87 | 87 | 1056 | 32 | 36 |
| $\% \text{ Of solved}$ | 96 | 66 | 70 | 95 | 95 | 95 | 71 | 74 | 77 |
(iii) It can be observed that the CPLEX is most effective for small instances; in the first stage 100% of test instances are solved (Table 3), in the second stage 97% (Table 4), and in the third stage 96% of test instances are solved (Table 5). However, for large test instances, MSBB-ADVRP is the best method (100% for the first and the second stages, 95% for the third stage).

6. Conclusions

In this paper we consider ADVRP and suggest exact algorithms for solving it. Our solution methods are based on relaxation to Assignment problem as it was first time proposed by Laporte et al. (1987). In order to rebuild feasibility, we branch by using tolerance criterion. We found that our simple method is fast but has the memory consumption problem. Therefore, we introduce the new method based on randomness in choosing the next node in the branch and bound tree and multistart.

Computational experiments show that with our multi–start approach (MSBB-ADVRP), we are able to solve at least 77% of instances in all stages with up to 1000 customers. In addition, our exact method is very efficient. For example in Table 5 it appears that MSBB-ADVRP on average needs 36 seconds for an instance, while CPLEX needs on average 1056 seconds. As far as we know, we are able to exactly solve the problems with larger dimension in comparison with the previous methods from the literature. For example, the largest problem solved by CPLEX has n = 360 nodes, while we are able to solve the problem exactly with n = 1000 nodes. The size of the problem we can solve depends on the available computer RAM memory. Thus, our approach may be even more effective and efficient in the future by using computers with larger memory.

We are working currently to improve the running computational times of the algorithm by trying to develop a good initial heuristic based on “first cluster then route” idea (Carrizosa et al., 2011), followed by Variable neighborhood search (Hansen et al., 2012a, b), and use it as initial upper bound. Another possibility is to improve lower bounds, which we do not explore in this paper, by adding more constraints to the assignment problem, or to relax some of them using Lagrangian multipliers. Such an approach does not use advantage provided by fast Hungarian method and could be research topic for the future work. Although we implement our B&B based method on DVRP problem, the method is quite general and may be adapted for solving other VRP variants.

References


