*Les Cahiers du GERAD* ISSN: 0711–2440

# Multistart Branch and Bound for Large Asymmetric Distance-Constrained Vehicle Routing Problem

S. Almoustafa, S. Hanafi,
N. Mladenović

G–2011–29

June 2011

# Multistart Branch and Bound for Large Asymmetric Distance-Constrained Vehicle Routing Problem

**Samira Almoustafa**

*Brunel University*
*Uxbridge, UB8 3PH, UK*
samira.almoustafa@brunel.ac.uk

**Said Hanafi**

*LAMIH*
*Université de Valenciennes, France*
said.hanafi@univ-valenciennes.fr

**Nenad Mladenović**

*GERAD & Brunel University*
*Uxbridge, UB8 3PH, UK*
nenad.mladenovic@brunel.ac.uk

June 2011

*Les Cahiers du GERAD*

G–2011–29

## Abstract

The asymmetric distance–constrained vehicle routing problem, consists of finding vehicle tours to connect all customers with a depot, such that the total distance is minimum and the traveled distance by each vehicle is less than or equal to the given maximum value. In order to solve it exactly we develop new tolerance based branching rules within branch and bound method. Since our method was fast but memory consuming, it could stop before optimality is proven. Therefore we introduce the randomness in choosing the node of the search tree. If an optimal solution is not found, we restart our procedure. In that way we get multistart branch and bound method. Computational experiments show that we are able to exactly solve large test instances with up to 1000 customers. As far as we know those instances are the largest compared with other approaches from the literature.

**Key Words:** Vehicle Routing Problem, Branch and Bound, Tolerance, CPLEX.

## Résumé

Le problème asymétrique de routage de véhicule avec contraintes de distance consiste à trouver des tournées de véhicules connectant l'ensemble des usagers avec un dépôt de telle sorte que la distance totale soit minimum et la distance parcourue par chaque véhicule plus petite ou égale à la valeur maximum donnée. Pour le résoudre exactement, nous développons de nouvelles règles de branchement basées sur la tolérance dans une méthode d'optimisation par séparation. Comme notre méthode est rapide mais gourmande en mémoire, elle peut être arrêtée avant que l'optimalité ne soit prouvée. En conséquence, nous introduisons la randomnisation dans le choix du sommet de l'arbre de recherche qui sera le suivant à être sélectionné. Si une solution optimale n'est pas trouvée, la méthode est réappliquée. De la sorte, nous optenons une méthode multi-start à l'intérieur d'une procédure de séparation. Des expériences de calculs montrent qu'avec notre recherche multi-start et d'optimisation par séparation nous arrivons à résoudre exactement de grandes instances avec jusqu'à 1000 usagers.

## 1 Introduction

The vehicle routing problem VRP is defined as follows: finding optimal tours 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) and Toth and Vigo (2002). 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 $j$ is different from node $j$ to node $i$, we call this problem asymmetric (ADVRP) otherwise the symmetric DVRP is defined. Surprisingly ADVRP is not studied as other types of VRPs. To the best of our knowledge, the last such article was published by Laporte et al. (1987).

Three types of algorithms are used to solve any VRP. The first type consists of exact algorithms. It has been studied by Lenstra and Rinnooy Kan (1975); Christofides et al. (1981); Laporte et al. (1985); Laporte and Nobert (1987), etc. This type of methods is time-consuming. The second type consists of classical *heuristics* such as greedy, local search, relaxation based etc. It has been also studied by many researchers, e.g., Clarke and Wright (1964); Lawer et al. (1985); Toth and Vigo (2002). They produce an approximate solution faster, when compared to the first type, but without guarantees of optimality. The third type consists of heuristics that are based on some *metaheuristic* rules. Such metaheuristics or framework for building heuristics are Simulated annealing, Tabu search, Genetic algorithms (Toth and Vigo, 2002), Variable neighborhood search (see recent surveys in Hansen et al. (2010), and Mladenović and Moreno-Pérez (2012)), etc.

In this paper we suggest a new exact algorithm for solving ADVRP that is based on Branch and Bound (B&B) method. The ADVRP problem is first transformed to the Travelling salesman problem (TSP). The lower bounds are obtained by relaxation of subtour elimination and maximum distance constraints. Thus the Assignment problem (AP) is solved in each node of the B&B tree. We use the best-first-search strategy and adapted tolerance based rules for branching. That is, the next node in the tree is one with the smallest relaxed objective function value. In the case of tie, we use two tie-breaking rules: (i) the last one in the list; (ii) the random one among them. We found that our B&B based method is very fast but memory consuming. That is why we suggested 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 structure of this paper is as follows. In Section 2 we present mathematical programming formulations of ADVRP. In Section 3 we discuss details of deterministic Branch and Bound method for ADVRP. In Section 4 we present our multistart approach for solving ADVRP. Section 5 contains details regarding data structure that we used in our implementation. Computational results are provided in Section 6. In Section 7 we give conclusion and future research directions.

## 2 Mathematical Programming Formulations for ADVRP

In this section we give two mathematical programming formulations of ADVRP. The first one, so-called flow based formulation, is used for comparison purposes in 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 that will be described in Section 3. See also recent reformulation of General VRP into the classical VRP in (Pop, 2011).

Let $N' = \{1, 2..., n-1\}$ denotes the set of customers and $V' = N' \cup \{0\}$ denotes the set of nodes where 0 is index of the depot. A set of arcs is denoted by $A'$, $A' = \{(i,j) \in V' \times V' : i \neq j\}$. The travelled distance from customer $i$ to customer $j$ and the number of vehicles are denoted by $d'_{ij}$ and $m$ respectively. The shortest distance between customer $i$ and customer $j$ is denoted by $c_{ij}$. The decision binary variable $x_{ij}$ is defined as follows:

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

**Flow based formulation.** For the sake of comparison, we use another formulation of `ADVRP` with polynomial number of variables and constraints, without copying depots. This is achieved by introducing the new set of variables $z_{ij}$. They present the shortest length travelled from the depot to customer $j$, where $i$ is the predecessor of $j$. The formulation of `ADVRP`, that will be later used with `CPLEX` solver (`CPLEX-ADVRP`), is given below (Kara, 2008):

$$f = \min \sum_{(i,j)\in A'} c_{ij}x_{ij} \tag{2}$$

subject to

$$\begin{aligned}
&\sum_{i\in N'} x_{ij} = 1 && \forall\ \ j \in N' && (3)\\
&\sum_{j\in N'} x_{ij} = 1 && \forall\ i \in N' && (4)\\
&\sum_{i\in N'} x_{i0} = m && && (5)\\
&\sum_{j\in N'} x_{0j} = m && && (6)\\
\sum_{(i,j)\in A'} z_{ij} - \sum_{(i,j)\in A'} z_{ji} - &\sum_{j\in V'} c_{ij}x_{ij} = 0 && \forall\ i \in N' && (7)\\
&z_{ij} \le (D_{max} - c_{j0})x_{ij} && \forall\ (i,j) \in A' && (8)\\
&z_{ij} \ge (c_{ij} + c_{0i})x_{ij} && \forall\ i \ne 0, \forall (i,j) \in A' && (9)\\
&z_{0i} = c_{0i}x_{0i} && \forall\ i \in N' && (10)\\
&x_{ij} \in \{0,1\} && \forall\ (i,j) \in A'. && (11)\\
& && && (12)
\end{aligned}$$

Obviously, there are polynomial number of variables and constraints. This model is known as flow based model since constraint (7) is typical flow constraint. It says that the distance from node $i$ to any other node $j$ on the tour should be equal to the difference between distance from depot to $i$ and distance from depot to $j$. This flow based formulation (2)–(11) gives better result than node based formulation as noted in Demir and Kara (2008). Constraint (8) presents that the total distance from depot to customer $j$ and the shortest distance from customer $j$ to depot directly are less than or equal to the maximum distance allowed. In addition, according to constraint (9) the total distance from depot to customer $j$ should be greater than or equal to the distance from the depot to customer $i$ plus the distance from customer $i$ to customer $j$. Constraint (10) gives the initial value for $z_{0i}$ which is equal to the distance from depot to customer $i$. Last constraint (11) introduces the decision variables $x_{ij}$ as binary variables.

**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

$$V = V^{'} \cup \{n, n+1, \ldots, n+m-2\}.$$

The distance matrix $D$ is obtained from $D'$ by the following transformation rules where $i,j \in V$:

$$d_{ij} = \begin{cases} d'_{ij} & \text{if } (0 \le i < n, 0 \le j < n, i \ne j)\\ d'_{0j} & \text{if } (i \ge n, 0 < j < n)\\ d'_{i0} & \text{if } (0 < i < n, j \ge n)\\ \infty & \text{otherwise} \end{cases}$$

Then the formulation of `TSP` is given below (13)–(16) as follows:

$$f(S) = \min \sum_{(i,j)\in A} d_{ij}x_{ij} \tag{13}$$

where $x_{ij}$ satisfies these conditions

$$\sum_i x_{ij} = 1 \qquad \forall\, j \in V \tag{14}$$

$$\sum_j x_{ij} = 1 \qquad \forall\, i \in V \tag{15}$$

$$\sum_{i,j \in U} x_{ij} \leq \mid U \mid -1 \qquad \forall\, U \subset V \backslash \{0\}, |U| \geq 2, \tag{16}$$

$$+ \text{ distance constraints} \tag{17}$$

The constraints (14) and (15) ensure that *in* and *out* degree of each node are equal to 1. The constraint (16) eliminates subtours (Dantzig et al., 1954), where $U$ is any subset of $V \backslash \{0\}$. To formulate ADVRP in addition to (13)–(16), we need to add distance constraint (17), which check that the total distance for each tour should be less than maximum distance allowed ($D_{max}$).

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

## 3 Branch and Bound for ADVRP

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

- The initial feasible solution of good quality is usually obtained by heuristic and its objective function value is initial upper bound ($UB$). If the heuristic solution is not known, then the upper bound is set to infinity ($UB = \infty$);
- The original problem is placed at the root of the branch and bound or search tree. All other nodes represent subproblems. In solving subproblems some variables are fixed and some constraints are ignored (relaxed), so that the lower bound is obtained;
- 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 Lower Bounds

To apply B&B we need to have lower bounding procedure that should be applied in 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 bound is obtained. We use AP as 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 its implementation described in Jonker and Volgenant (1986). The complexity of AP at root node is in $O(n^3)$ (Volgenant, 2006). Another advantage is that the relaxed AP solution is already integer.

**Proposition 1.** *Any feasible* AP *solutions for 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 each cycle* $\omega \geq 2$.

*Proof.* It is clear from (14) and (15) that the degree of each vertex in $S$ is equal to 2. It has one incoming and one outcoming arc. If the matrix $D$ was symmetric, then those cycles would contain just 2 vertices and therefor $\omega = 2$. However, in asymmetric case, $\omega$ could be obviously larger than 2: if vertex $i$ is assigned to $j$, then $j$ is not neccessary assigned to $i$. □

There are 3 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 or 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 $D_{max}$ or if it contains no depot.

## 3.2 Tolerance Based 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 to it large value ($\infty$) and then resolve `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 `AP` 's.

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 Koltai 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 obtained. Therefore, in our `ADVRP` tolerance based `B&B` (`TOL-ADVRP`), the arc which has the smallest upper tolerance is chosen to be excluded. Some preliminary results of this approach has been given in Almoustafa et al. (2009).

Another possibility of destroying infeasible tour is to exclude the arc with the largest cost. `B&B` method that uses such a branching rule we call `COST-ADVRP`. However, based on extensive computational analysis, the results obtained with `COST-ADVRP` were of slightly worse quality than those obtained by our `TOL-ADVRP`. That was the reason why in computational analysis section we give `TOL-ADVRP` results. In `TOL-ADVRP` when there are more than one subproblem in the list of active subproblems (that have the smallest objective value) we choose the last among them to branch next. In other words, our tie-breaking rule is deterministic.

## 3.3 Algorithm

The main steps of `TOL-ADVRP` algorithm are given in Algorithm 1, where we use the following notation:

$S$ − the relaxed solution obtained by `AP` or $S = \{T_1, T_2, ..., T_{M(S)}\}$ presented as a set of tours where $M(S)$ is the number of tours in $S$;

$d(T_k)$ − the total travelled distance in a tour $k$ where $d(T_k) = \sum_{(i,j)\in T_k} d_{ij}$;

$t(T_k)$ − the number of arcs in a tour $k$;

$f(S) = \sum_{k=1}^{M(s)} d(T_k)$ − the value of an optimal solution to `AP`;

$S^*$ − an optimal solution to `ADVRP`;

$L$ − the list of active subproblems (or unfathomed nodes in a search tree), it is updated during the execution of the code;

$APcnt$ − counts the number of nodes in `B&B` tree (how many times `AP` subroutine is called);

$Maxnodes$ − the maximum number of nodes allowed in `B&B` tree. In order to prevent termination with no memory message, we use it as stopping condition. Here we set $Maxnodes = 100,000$;

$\beta-$ the type of tie-breaking rule that has two values:

$$\beta = \begin{cases} 0 & \text{deterministic (last in } L\text{);} \\ 1 & \text{at random.} \end{cases}$$

*ind* $-$ the variable that covers all possible outputs of the algorithm. The basic algorithm may stop with the following outputs:

$$ind = \begin{cases} 1 & \text{an optimal solution } S^* \text{ is found ;} \\ 2 & \text{feasible solution } S \text{ is found but not proven as optimal;} \\ 3 & \text{no feasible solution is found (lack of memory);} \\ 4 & \text{there is no feasible solution of the problem at all.} \end{cases}$$

```
   Procedure: TOL-ADVRP($n, m, D_{max}, D, Maxnodes, \beta, S^*, ind$);
 1 $UB \leftarrow m \times D_{max}$, $APcnt \leftarrow 1$, set iteration counter $i \leftarrow 1$;
 2 Solve AP to get its solution $S$ and find $\Gamma(S) = \{T_l | l = 1, ..., M\}$;
 3 $L = \{1\}$- the list contains the root node;
 4 Calculate $d(T_l)$ for every tour of $T_l \in \Gamma(S)$;
 5 if ($S$ feasible) then ($S^* = S$ is an optimal solution; $ind$=1; stop);
   while ($APcnt < Maxnodes$) do
 6     Branching. Choose subproblem $b_i \in L$ with the smallest value of the
       objective function; in the case of tie, choose one from the list with respect
       to $\beta$ value;
 7     Best first. Find the ratio $d(T_k)/t(T_k)$ for every infeasible tour $k =
       1, .., M'$ where $M'$ is the number of infeasible tours and choose the tour
       $k^*$ with the largest ratio;
 8     Tolerance (expanding search tree). Calculate upper tolerances for all
       arcs in this $k^*$ tour by solving $t(T_{k^*})$ times AP problem to get solutions $S_r$
       where $r = 1, ..., t(T_{k^*})$. Expand search tree with those $t(T_{k^*})$ subproblems
       and update $APcnt$ as: $APcnt = APcnt + t(T_{k^*})$;
 9     Feasibility check. Check feasibility of all new (expanded) nodes;
10     Update. Update $UB$ (if necessary), and $L$ based on the current $UB$
       value as: $L \leftarrow \{r | f(S_r) < UB\} \setminus \bigcup \{b_j | j = 1, .., i\}$ where $S_r$ are infeasible
       solutions;
11     Optimality conditions. If ($L = \emptyset$ and $UB \neq m \times D_{max}$) then ($S^*$
       is the optimal solution where $f(S^*) = UB$; $ind$=1; stop). Otherwise, If
       ($L = \emptyset$ and $UB = m \times D_{max}$) then (there is no feasible solution; $ind$=4;
       stop);
12     $i = i + 1$;
   end
13 Termination. If ($UB \neq m \times D$) then ($S$ is the new incumbent; $ind$ =2;
   return), otherwise (no memory; $ind$=3; return) ;
```

**Algorithm 1**: (TOL-ADVRP) algorithm with tolerance based branching

Here we explain some of steps of algorithm TOL-ADVRP: At the root node we find solution $S$ by solving AP problem. Then we calculate the total distance for every tour of $S$ and check the feasibility of the solution. If it is feasible, then the optimal solution exists and the program stops. Otherwise, we repeat the following steps until the memory limit is reached:

- **Branching.** Choose the subproblem $S \in L$ with the smallest value of the objective function. In the case when more than one subproblem has the smallest value, the choice is made according to vaue of $\beta$. If $\beta = 0$ we choose last subproblem in the list $L$; otherwise choose one randomly to develop further. This second option will be used after in our multistart method.
- **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. Choose the tour with the largest ratio to branch.

- **Tolerance (expanding search tree).** Calculate upper tolerances for all arcs in this tour as follows: Exclude in turn one arc from this tour by putting $\infty$ in the distance matrix. Find $AP$ solution to those subproblems. Check feasibility for each new subproblem. If this subproblem produces a feasible solution $S$, and its value is smaller than $UB$ then update the value of the upper bound ($UB = f(S)$). Find the difference between the value of the objective function before and after excluding the arc. This gives the value of upper tolerance ($UT$) to this arc. Note that the value of the counter $APcnt$ is increased by the number of arcs in the chosen tour:

$$APcnt = APcnt + t(T_{k^*})$$

- **Check Feasibility.** Check feasibility of the solution at every new subproblem which the program generates,
- **Update.** If a feasible solution is found and its value is smaller than current upper bound then update the value of upper bound, update the list of active subproblems by removing the subproblems that have value greater than the value of upper bound and by adding the new expanded subproblems that has value smaller than $UB$ as follows:

$$L \leftarrow \{r|f(S_r) < UB\} \backslash \bigcup \{b_j|j = 1, .., i\}$$

  Note $S_r$ is infeasible solutions.
- **Optimality conditions.** Check if $L = \emptyset$ and $UB$ is updated then stop with the value of optimal solution $f(S^*) = UB$ ($ind$=1). Otherwise, if $L = \emptyset$ and $UB$ is not updated, stop with the message that no feasible solution exists ($ind$=4),
- **Termination.** When there in no memory we get two possible outputs: If $UB$ is updated then $S^*$ is returned as a feasible but not proved as optimal solution ($ind$=2). Otherwise, a feasible solution has not been found, but that does not mean the feasible solution does not exist ($ind$=3).

**Proposition 2.** *Algorithm* `TOL-ADVRP` *finds an optimal solution to* `ADVRP` *or proves that such a solution does not exist.*

*Proof.* It is enough to show that our `B&B` algorithm enumerates all possible `VRP` tours with $m$ vehicles (assuming that $D_{max} = \infty$ and there is no memory restriction). Really, our enumeration is based on eliminating arcs by giving them value $\infty$ (in step 8). 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 generated with our method contains all possible cycles with $m$ vehicles. On the other hand, the set of all feasible `ADVRP` tours are the subset of all this generated set. Therefore, our `TOL-ADVRP`, enumerates all feasible tours. □

## 3.4 Illustrative Example

The example from Table 1 is taken from Balas and Toth (1985), where the number of customers is $n = 8$ and the number of vehicles is $m = 2$. The location of the first customer is considered as a depot.

Table 1: Original distance matrix for `ADVRP` with $n$=8 and $m$=2

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | $\infty$ | 2 | 11 | 10 | 8 | 7 | 6 | 5 |
| 1 | 6 | $\infty$ | 1 | 8 | 8 | 4 | 6 | 7 |
| 2 | 5 | 12 | $\infty$ | 11 | 8 | 12 | 3 | 11 |
| 3 | 11 | 9 | 10 | $\infty$ | 1 | 9 | 8 | 10 |
| 4 | 11 | 11 | 9 | 4 | $\infty$ | 2 | 10 | 9 |
| 5 | 12 | 8 | 5 | 2 | 11 | $\infty$ | 11 | 9 |
| 6 | 10 | 11 | 12 | 10 | 9 | 12 | $\infty$ | 3 |
| 7 | 7 | 10 | 10 | 10 | 6 | 3 | 1 | $\infty$ |

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 remaining customers. To solve this problem, we have to add $m - 1 = 2 - 1 = 1$ copy of a 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), where 0 and 8 represent two depots for this problem. We will consider 2 problems with this dataset with 2 different values of $D_{max}$. First value of $D_{max(1)}$ is $\infty$, which produce as output the longest tour $LT$ in the optimal solution, then the second value of maximum distance allowed $D_{max(2)}$ is chosen to be $0.90 \times LT$. In addition, we will use $\overline{f_b}(i,j)$ as the value of `AP` at the subproblem $b$ with $d(i,j) = \infty$;

Table 2: New distance matrix

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | $\infty$ | 2 | 11 | 10 | 8 | 7 | 6 | 5 | $\infty$ |
| 1 | 6 | $\infty$ | 1 | 8 | 8 | 4 | 6 | 7 | 6 |
| 2 | 5 | 12 | $\infty$ | 11 | 8 | 12 | 3 | 11 | 5 |
| 3 | 11 | 9 | 10 | $\infty$ | 1 | 9 | 8 | 10 | 11 |
| 4 | 11 | 11 | 9 | 4 | $\infty$ | 2 | 10 | 9 | 11 |
| 5 | 12 | 8 | 5 | 2 | 11 | $\infty$ | 11 | 9 | 12 |
| 6 | 10 | 11 | 12 | 10 | 9 | 12 | $\infty$ | 3 | 10 |
| 7 | 7 | 10 | 10 | 10 | 6 | 3 | 1 | $\infty$ | 7 |
| 8 | $\infty$ | 2 | 11 | 10 | 8 | 7 | 6 | 5 | $\infty$ |

$LB$ solution at the root node of `B&B` tree, obtained by solving `AP`, is given in Figure 1:a. Each depot label is written inside a squared box and the total distance is written inside each tour. The value of the objective function to the relaxed problem at the root node, obtained by solving `AP`, is $f_1 = 29$ (see Figure 1:a).

**Problem 1 ($D_{max(1)} = \infty$).** We consider the problem with $D_{max} = \infty$. we set $APcnt = 1$, $L = \{1\}$, $i = 1$, and $b_1 = 1$. Clearly, initial value of $UB = m \times D_{max} = \infty$.

***Iteration 1.*** It can be seen from Figure 1:a that we obtain solution $S$ with three tours: one of them is infeasible, because it does not contain depot. Therefore this solution $S$ is not feasible.

The program will check the total distance for all tours in the solution at the root node $d(T_m) = \{d(T_1), d(T_2), d(T_3)\} = \{16, 8, 5\}$. Since only the last tour $T_3$ is *infeasible*, there is one possible subtour for branching. The number of arcs in $T_3$ is equal to 3 ($t(T_3) = 3$) and its total distance is equal to 5 ($d(T_3) = 5$).

The value of upper tolerance for every arc $\{(5,3), (3,4), (4,5)\}$ in the chosen tour $T_3$ is calculated as follows:

**(i) Arc (5,3)**: exclude this arc from the solution by putting that its length is equal to $\infty$ instead of its original value $d(5,3) = 2$ in the new distance matrix (Table 2). Then find the solution of `AP` with $d(5,3) = \infty$. This solution is not feasible for `ADVRP` (Figure 1:b) because there is at least one infeasible tour (in fact there are two infeasible tours $T_3, T_4$).
The value of the optimal `AP` solution for the new distance matrix at node 2 in the search tree is $f_2 = \overline{f}_1(5,3) = 34$.
The upper tolerance value (UT) for the arc $(5,3)$ in the optimal solution $S$ is calculated as :

$$UT(5,3) = \overline{f}_1(5,3) - f_1 = 34 - 29 = 5$$

**(ii) Arc (3,4)**: first restore $d(5,3)$ into its previous value 2. By excluding an arc $(3,4)$ as before, we get $f_3 = \overline{f}_1(3,4) = 35$. The solution at node 3 is also not feasible for `ADVRP` since it contains one infeasible tour $T_3$ (Figure 1:c). The value of the upper tolerance value for arc $(3,4)$ is :

$$UT(3,4) = \overline{f}_1(3,4) - f_1 = 35 - 29 = 6$$

**(iii) Arc (4,5)**: as before restore $d(3,4)$ into its previous value 1. Excluding an arc $(4,5)$ produces $f_4 = \overline{f}_1(4,5) = 33$ (Figure 1:d). The solution at node 4 is not a feasible solution because it contains one infeasible tour $T_3$. The value of upper tolerance for arc $(4,5)$ is :

$$UT(4,5) = \overline{f}_1(4,5) - f_1 = 33 - 29 = 4$$

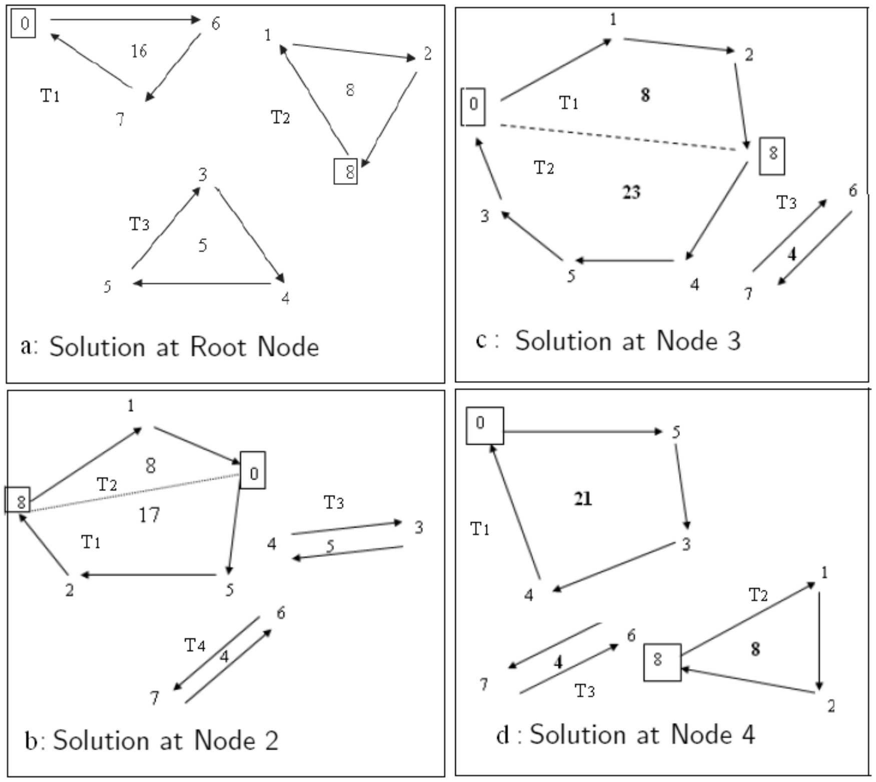


Figure 1: Solutions at root node (Node 1), Node 2, Node 3, and Node 4

The value of $APcnt = 1 + 3 = 4$. No need to update $UB$ Since no feasible solution is found, while we need to update $L$. The list of active subproblems $L = \{2, 3, 4\}$.

***Iteration 2.*** The smallest upper tolerance is at node $b = 4$. Therefore, the arc $(4, 5)$ is excluded and $LB = f_4 = 33$. We start now from subproblem 4, which gives an infeasible solution (Figure 1:d). Among 3 tours in that solution $S$ only one is infeasible. It has two arcs. Thus, in this case, the two new subproblems are generated by the program:

$$f_5 = \overline{f}_4(7, 6) = 34 \text{ (feasible solution);}$$
$$f_6 = \overline{f}_4(6, 7) = 37 \text{ (infeasible solution).}$$

Update the value of upper bound $UB$ ($UB = 34$) and the list of active subproblems ($L = \emptyset$) and the counter $APcnt = 6$. So the optimal solution is found at node 5 in the search tree (Figure 2:b). In this small example, the total number of subproblems generated in the search tree is 6.

**Problem 2 ($D_{max(2)} = 23$).** The longest tour in the optimal solution is 26, we use this value to get new value to $D_{max}$ where $D_{max} = 0.9 \times 26 = 23.4$. We will consider only the integral part of this number so that the new value of maximum distance allowed is $D_{max} = 23$. We update the value of $D_{max}$, and we run the same example.

***Iteration 1.*** It will be the same as iteration 1 in problem 1, i.e., $L = \{2, 3, 4\}$.

***Iteration 2.*** In this iteration we have $L = \{2, 3, 5, 6\}$. Moreover we have two nodes (node 2 and node 5) that have the same smallest value 34 (Figure 2:b). Therefore, according to tie-breaking rule for `TOL-ADVRP`, we have to choose the last node to branch next ($b_2 = 5$). When we branch at node 5, we find first feasible solution at node 9, so we update $UB = 37$, $APcnt = 12$, $L = \{2, 3, 6, 7, 10, 12\}$.

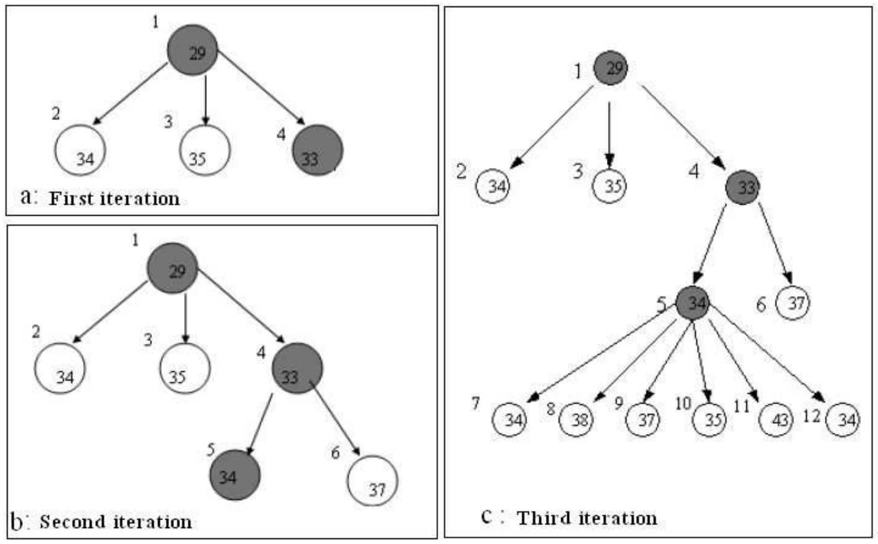


Figure 2: TOL-ADVRP Tree

***Iteration 3.*** In this iteration we have 3 nodes (2, 7, and 12) that have the same smallest value 34 (Figure 2:c), we have to choose the last node to branch next i.e., $b = 12$. etc.

At the end, after generating 58 subproblems ($APcnt = 58$) we get optimal solution value 37.

## 4 Multistart Method

As mention earlier, motivation for developing multistart B&B method for solving ADVRP is based on the fact that TOL-ADVRP is very fast but requires a lot of computer memory. It usually stops after a few seconds reaching the maximum number of nodes visited in the search tree. The main idea of (MSBB-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 exact B&B method hoping that in the next attempt we will get an optimal solution. For that reason we rerun TOL-ADVRP (with $\beta = 1$) many times until an optimal solution is found or infeasibility is proven. Therefore, the (MSBB-ADVRP) may be summarized as follows:

1. Re-run each instance given number of times ($ntrail$- a parameter); stop when an optimal solution is found ($ind = 1$) or infeasibility of the problem instance is proved ($ind = 4$). This will increase the chance to find an optimal solution or at least improve the value of the best feasible solution found so far;
2. Choose randomly one subproblem from the list of active subproblems ($L$) as follows:
   - Generate uniform random number $\alpha \in [0, 1]$;
   - Find the number of nodes in the list ($ns$) which has value equal to the smallest objective function value in this list;
   - Find $k \in [1, ns]$ as $k = 1 + ns * \alpha$;
   - Branch on the node corresponding to $k^{th}$ position in the list $L$.

### 4.1 Algorithm

### 4.2 Illustrative Example

We now present our MSBB-ADVRP on the same example from the previous section. We do not consider Problem 1, since there were no ties.

```
   Algorithm: MSBB-ADVRP($n, m, D_{max}, D, Maxnodes, \beta, ntrail, S_{best}$);
1  $f_{best} \leftarrow \infty$, $\beta \leftarrow 1$;
   for $i = 1$ to $ntrail$ do
2      TOL-ADVRP($n, m, D_{max}, D, Maxnodes, \beta, S^*, ind$);
3      if($ind = 1$)then ($S_{best} = S^*$; stop);
4      if($ind = 4$)(stop);
5      if $1 < ind < 4$ then
6          if $(f(S^*) < f_{best})$ then
               $S_{best} = S^*$, $f_{best} = f(S^*)$;
           end
       end
   end
```

**Algorithm 2**: Algorithm of Multistart B&B (MSBB-ADVRP)

**Problem 2.** We run the example with $D_{max(2)} = 23$. In the first iteration both programs do the same because we have only one smallest value in $L$ (see Figure 2:a). At the second iteration two nodes 2 and 5 have the same smallest value of 34 (Figure 2:b). So, there are two options. Suppose we randomly choose node 5. Then we get the solution as given at Figure 2:c. In the third iteration we have the number of subproblems with the same value is equal to 3 ($ns = 3$). So we need to choose one $k$ among three nodes: 2, 7 or 12 where $k = 1 + 3\alpha$. Assume that $\alpha = 0.4$, then $k = 2$ and the second node (7) is chosen for branching. This step is repeated at each iteration until the optimal solution is found.

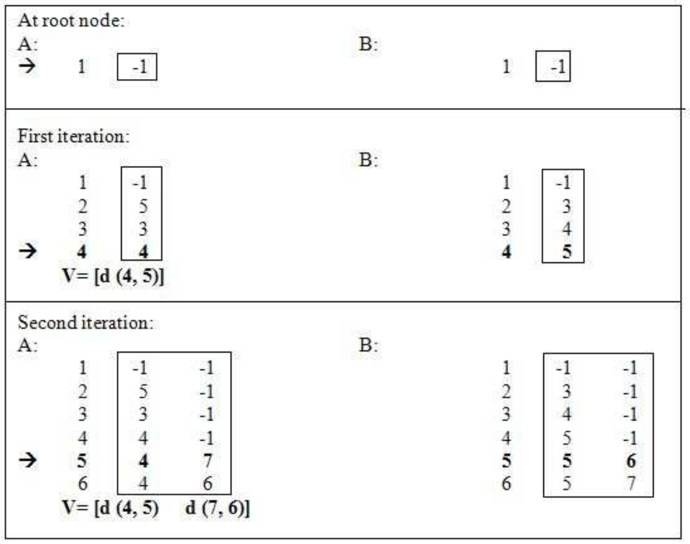
## 5 An Efficient Implementation – Data Structure

The most important task in implementation of TOL-ADVRP is to keep track of excluded arcs $(a, b)$ during complete enumeration within B&B tree. Left-hand side arcs $(a)$ are stored in the first matrix $A$ and the right-hand side arcs $(b)$ are stored in the second matrix $B$. Those matrices are expanded during the execution of the code. Each row of $A$ and $B$ represents a node in the search tree. Thus, we always start with $A = [-1]$ and $B = [-1]$ since there are no excluded arcs at the root node. Note that we use symbol $(-1)$ to denote a dummy vertex. Each time some arc is excluded (its value set to $\infty$ and AP solved again), new row in both matrices are added, containing end points of excluded arc. In addition, each iteration brings a new column, where rows from previous iteration are filled with dummy vertices. Of course based on best-first-search criterion and tie-breaking rule, new rows of $A$ and $B$ (in new iteration) will keep track of excluded parent vertices from previous iterations. We will now present our data structure on the same example from the previous section. At the root node we have $A = [-1]$, $B = [-1]$ (see Figure 3).

At the first iteration we have 3 new nodes in the search tree, or 3 possible arcs to be excluded (see Figure 2:a). So we add 3 more rows to our matrices $A$ and $B$. Thus, we have only one column and 4 rows in both matrices. At node 2 we exclude arc $(5, 3)$. So we put 5 in the row 2 of $A$, and 3 in the row 2 of $B$. The same is done for other two nodes (see Figure 3: First iteration). Node 4 is chosen to branch since the AB solution , after excluding arc $(4, 5)$, was the smallest. So we copy all the numbers from both matrices of node 4 and put them as initial values to all sons of node 4.

The sons of each parent node have identical rows except the last value. The number of values in each row which is not equal (-1) represents the number of excluded arcs in this node. We use temporary vector $(V)$ to save the values of excluded arcs in the node before we solve AP.

In the second iteration we have 2 more nodes. We add two rows to both matrices (exclude two arcs from some nodes), and one column. According to our example we choose node 4 to branch. The new two nodes will copy the information from the row of node 4. Then it will add the new arc that was excluded in each case (see Figure 3: Second iteration). For the root node all the row contains (-1), and for nodes (2,3,4) we put (-1) in the second column because we have excluded only one arc for them. Each time we increase the dimension of both matrix we have to put (-1) for all the previous nodes in the new columns.



At root node:

A:

| | | |
|---|---|---|
| → | 1 | -1 |

B:

| | |
|---|---|
| 1 | -1 |

First iteration:

A:

| | | |
|---|---|---|
| | 1 | -1 |
| | 2 | 5 |
| | 3 | 3 |
| → | **4** | **4** |

V= [d (4, 5)]

B:

| | |
|---|---|
| 1 | -1 |
| 2 | 3 |
| 3 | 4 |
| **4** | **5** |

Second iteration:

A:

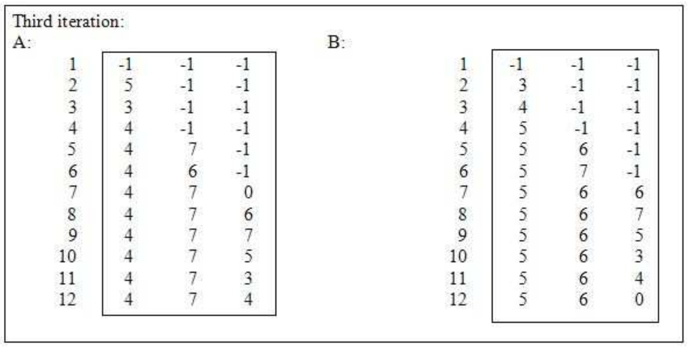
| | | | |
|---|---|---|---|
| | 1 | -1 | -1 |
| | 2 | 5 | -1 |
| | 3 | 3 | -1 |
| | 4 | 4 | -1 |
| → | **5** | **4** | **7** |
| | 6 | 4 | 6 |

V= [d (4, 5) d (7, 6)]

B:

| | | |
|---|---|---|
| 1 | -1 | -1 |
| 2 | 3 | -1 |
| 3 | 4 | -1 |
| 4 | 5 | -1 |
| **5** | **5** | **6** |
| 6 | 5 | 7 |

Figure 3: First two iterations



Third iteration:

A:

| | | | |
|---|---|---|---|
| 1 | -1 | -1 | -1 |
| 2 | 5 | -1 | -1 |
| 3 | 3 | -1 | -1 |
| 4 | 4 | -1 | -1 |
| 5 | 4 | 7 | -1 |
| 6 | 4 | 6 | -1 |
| 7 | 4 | 7 | 0 |
| 8 | 4 | 7 | 6 |
| 9 | 4 | 7 | 7 |
| 10 | 4 | 7 | 5 |
| 11 | 4 | 7 | 3 |
| 12 | 4 | 7 | 4 |

B:

| | | | |
|---|---|---|---|
| 1 | -1 | -1 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 4 | -1 | -1 |
| 4 | 5 | -1 | -1 |
| 5 | 5 | 6 | -1 |
| 6 | 5 | 7 | -1 |
| 7 | 5 | 6 | 6 |
| 8 | 5 | 6 | 7 |
| 9 | 5 | 6 | 5 |
| 10 | 5 | 6 | 3 |
| 11 | 5 | 6 | 4 |
| 12 | 5 | 6 | 0 |

Figure 4: Third iteration

In the third iteration of branching we choose to branch on node 5. This will produce 6 new nodes. So we need to add 6 rows and one more column to both matrices (see Figure 4: Third iteration).

## 6 Computational Results

**Computers:** All experiments were implemented under windows XP and on intel(R) Core(TM)2 CPU 6600@2.40GHz, with 3.24 GB of RAM. The code is written in C++ language. Some parts of the code are taken from publicly available source code (Turkensteen, 2007).

**Test instances:** A 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; 0.5 means 50% symmetric, etc;

*seed* – the random number $\in (0,1)$: when $n \leq 200$, four different seeds were chosen to generate four different distance matrices for each combination of $(n,m)$. However, only one distance matrix is generated in case: $200 < n \leq 1000$.

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 (with $n = 40, 60, .., 200$) and large size (with $n = 240, 280, ..., 1000$). For each $n$ belonging to the small set, two different types of instances are generated, based on the different number of vehicles: $m_1 = n/20$ and $m_2 = n/10$. For instances belonging to the large set, we use only $m_1$. In addition, for each distance matrix we consider 3 problems with 3 different values of $D_{max}$ (see Section 3.4). First value of $D_{max}$ is $\infty$, then we use this formulae to obtain new value of $D_{max}$ :

$$D_{max(i)} = 0.90 \times LT(i-1)$$

Where $i \in \{2,3\}$, and $LT(i-1)$ is the longest tour in the optimal solution when the value of maximum distance allowed is $D_{max(i-1)}$. 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/index.htm` as well as the code for generator coded in C++.

**Methods compared.** In this article we compare three methods to find an optimal solution to `ADVRP`: `MSBB-ADVRP`, `TOL-ADVRP`, `CPLEX-ADVRP`. In all our experiments reported below, we run `MSBB-ADVRP` only two times. We note that increasing the number of restarts might improve chances to find an optimal solution, but with the cost of larger CPU time. In `CPLEX-ADVRP` the process will continue until an optimal solution is found or the time limit is reached. We choose the time limit to be 10800 seconds (3 hours) for all test instances.

**Comparison.** Tables 3, 4, and 5 contain summary results to all 257 test instances from $n = 40$ up to $n = 1000$ customers with $D_{max(1)} = \infty$, $D_{max(2)} = 0.90 \times LT(1)$, and $D_{max(3)} = 0.90 \times LT(2)$ respectively. The rows in all tables give the following characteristics:

1. $\sharp$ Opt $(ind = 1)$ – how many times each program finds an optimal solution;
2. $\sharp$ Feas $(ind = 2)$ – how many times feasible (but not optimal) solutions have been found. It presented as $a(b)$ where $a$ ( or $b$) is the number of times the feasible solutions (or the same value as optimal) has been found;
3. $\sharp$ No Mem $(ind = 3)$ – how many times feasible solutions are not found because of lack of memory;
4. $\sharp$ No Feas $(ind = 4)$ – how many instances with proven infeasibility are detected;
5. Total time – the total time spent only for instances where the optimal solutions are found;
6. Average time – the average time for all instances, where (Average time= Total time/$\sharp$ Opt);

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

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 $D_{max(1)} = \infty$), the rate of success are 100% for `TOL-ADVRP`, 80% for `CPLEX-ADVRP`, and 100% for `MSBB-ADVRP`. For the second stage (with $D_{max(2)} = 0.90 \times LT(1)$) the rate of success are 69%, 78% , and 79% for `TOL-ADVRP`, `CPLEX-ADVRP`, and `MSBB-ADVRP` respectively. Finally, in the third stage (with $D_{max(3)} = 0.90 \times LT(2)$) the rate of success for the three programs `TOL-ADVRP`, `CPLEX-ADVRP`, and `MSBB-ADVRP` are 55%, 71%, and 75% respectively.

**(ii)** Regarding efficiency, it can be seen that `TOL-ADVRP` and `MSBB-ADVRP` are much faster than `CPLEX-ADVRP`. The `MSBB-ADVRP` is more efficient than `TOL-ADVRP` for solving instances in the first two stages (total time for `TOL-ADVRP` in the first two stages are 4.61 and 959.53 seconds, and for `MSBB-ADVRP` are 4.31 and 262.14 seconds), while the opposite holds for the third stage (1302.76 seconds for `TOL-ADVRP`, and 1826.47 seconds for `MSBB-ADVRP`). However in that stage the number of instances solved exactly by `MSBB-ADVRP` and `TOL-ADVRP` are 55 and 40 respectively;

Table 3: Results for 92 instances with $D_{max(1)} = \infty$

| | 72 small test inst. | | | 20 larg test inst. | | | Total number. | | |
|---|---|---|---|---|---|---|---|---|---|
| | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` |
| ♯Opt $(ind = 1)$ | 72 | 72 | 72 | 20 | 2 | 20 | 92 | 74 | 92 |
| ♯Feas $(ind = 2)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ♯No Mem $(ind = 3)$ | 0 | 0 | 0 | 0 | 18 | 0 | 0 | 18 | 0 |
| ♯ No Feas $(ind = 4)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Total time | 0.72 | 21896.33 | 0.71 | 3.89 | 3766.31 | 3.6 | 4.61 | 25662.64 | 4.31 |
| Average time | 0.01 | 304.12 | 0.01 | 0.19 | 1883.16 | 0.18 | 0.05 | 346.79 | 0.05 |
| % of solved | 100 | 100 | 100 | 100 | 10 | 100 | 100 | 80.43 | 100 |

Table 4: Results for 92 instances with $D_{max(2)} = 0.90 \times LT(1)$

| | 72 small test inst. | | | 20 larg test inst. | | | Total number. | | |
|---|---|---|---|---|---|---|---|---|---|
| | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` |
| ♯Opt $(ind = 1)$ | 45 | 70 | 53 | 19 | 2 | 20 | 64 | 72 | 73 |
| ♯Feas $(ind = 2)$ | 19(11) | 0 | 16(14) | 0 | 2(0) | 0 | 19(11) | 2(0) | 16(14) |
| ♯No Mem $(ind = 3)$ | 6 | 0 | 1 | 1 | 16 | 0 | 7 | 16 | 1 |
| ♯No Feas $(ind = 4)$ | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 2 | 2 |
| Total time | 13.9 | 40972 | 141.65 | 945.5 | 12671 | 120.49 | 959.53 | 53644 | 262.14 |
| Average time | 0.30 | 569.06 | 2.58 | 49.76 | 6335.93 | 6.02 | 14.54 | 724.93 | 3.50 |
| % of solved | 65.28 | 100.00 | 76.39 | 95.00 | 10.00 | 100.00 | 69.57 | 78.26 | 79.35 |

Table 5: Results for 73 instances with $D_{max(3)} = 0.90\ of\ LT(2)$

| | 53 small test inst. | | | 20 larg test inst. | | | Total number. | | |
|---|---|---|---|---|---|---|---|---|---|
| | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` | `TOL` | `CPLEX` | `MSBB` |
| ♯Opt $(ind = 1)$ | 26 | 51 | 36 | 14 | 1 | 19 | 40 | 52 | 55 |
| ♯Feas $(ind = 2)$ | 16(5) | 2(0) | 14(13) | 1(0) | 0 | 0 | 17(5) | 2(0) | 14(13) |
| ♯No Mem $(ind = 3)$ | 11 | 0 | 3 | 5 | 19 | 1 | 16 | 19 | 4 |
| ♯ No Feas $(ind = 4)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Total time | 42.07 | 50869 | 206.17 | 1260.69 | 4032 | 1620.3 | 1302.76 | 54902 | 1826.47 |
| Average time | 1.62 | 997.45 | 5.73 | 90.05 | 4032.28 | 85.28 | 32.56 | 1055.80 | 33.20 |
| % of solved | 49.06 | 96.23 | 67.92 | 70.00 | 5.00 | 95.00 | 54.79 | 71.23 | 75.34 |

**(iii)** When compare small and large test instances, it can be concluded that the `CPLEX` is most effective for small instances in the second stage 100% of test instances are solved (Table 4) and in the third stage 96.23% of test instances are solved (Table 5). However, for large test instances, `MSBB-ADVRP` is the most effective (100% for the first and the second stage, 95% for the third stage);

**(iv)** Regarding average time for small instances, the most efficient method is `TOL-ADVRP` (0.01, 0.30, and 1.62 seconds in the first, the second, and the third stage), while for large instances `MSBB-ADVRP` is the most efficient: it takes 0.18, 6.02, and 85.28 seconds in the first, the second, and the third stage respectively. However, the most efficient on average is `MSBB-ADVRP`.

## 7 Conclusions

We consider `ADVRP` and suggest exact algorithm for solving it. They are based on Assignment Problem relaxation, as first time proposed by Laporte et al. (1987). In order to rebuild feasibility, we branch by using tolerance criterion. We found that our method is fast but it 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.

Computational experiments show that with our multistart approach we are able to solve at least 75% of instances in all stages with up to 1000 customers. It appears that `MSBB-ADVRP` on average needs 0.04 seconds for the first stage, 3.59 seconds for the second stage and 33.20 seconds for the third stage, while `CPLEX` needs on average 346.79, 724.93, 1055.81 seconds for the first, the second, and the third stage respectively. In summary, the results of experiments emphasize that using `MSBB-ADVRP` provides good solutions in reasonable computational time. Moreover, as far as we know, we are able to exactly solve problems with larger dimension than previous methods from the literature. For example the largest problem solved by `CPLEX` has $n = 360$ customers, while we are able to solve exactly with $n = 1000$ customers. It is interesting to note that the dimension of problems solved by our methods depends on available memory of the computer used. Thus, our approach may be used in the future using new computers having larger memory.

We are working currently to improve the running computational times of the algorithm by trying to develop a good heuristic such as Variable Neighborhood Search (Mladenović and Hansen, 1997; Hansen et al., 2010), and to use it as initial upper bound. Another possibility is to improve lower bounds that 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.

## References

Almoustafa, S., Goldengorin, B., Tso, M., Mladenović, N., 2009. Two new exact methods for asymmetric distance-constrained vehicle routing problem. Proceedings of SYM-OP-IS, 297–300.

Balas, E., Toth, P., 1985. Branch and bound methods. In: Lawer et al.,(eds) The traveling salesman problem. Wiley-interscience series in discrete mathematics. Chichester : Wiley, 361–401.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega. 34(3), 209–219.

Clarke, G., Wright, J. V., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research. 12(4), 568–581.

Clausen, J., 1999. Branch and bound algorithms-principles and examples. Department of Computer Science, University of Copenhagen, Denmark. World Wide Web, `http://www.imada.sdu.dk/Courses/DM85/TSPtext.pdf`.

Climer, S., Zhang, W., 2006. Cut and solve: An iteration search strategy for combinatorial optimization. Journal of Artificial Intelligence. 170, 714–738.

Christofides, N., Mingozzi, A., Toth, P., 1981. State space relaxation procedures for the comuptation of bounds to routing problems. Networks. 11(2), 145–164.

Dantzig, G.B., Fulkerson, D.R., Johnson, S.M., 1954. Solution of a large-scale traveling salesman problem. Operations Research. 2, 393–410.

Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. Management Science. 6(1), 80–91.

Dell'Amico, M., Righini, G., Salani, M., 2006. A branch-and-price approach to the vihcle routing problem with simultaneous distribution and collection. Transportation Science. 40(2), 235–247.

Demir, E., Kara, I., 2008. Formulations for school bus routing problems. In: 21st Conference on Combinatorial Optimization, Dubrovnik, Croatia, 29–31.

Golden, B., Raghavan, S., Wasil, E., 2008. The vehicle routing problem: latest advances and new challenges. Operations Research/Computer Science Interfaces Series; 43. New York : Springer.

Goldengorin, B., Ghosh, D., Sierksma, G., 2004. Branch and peg algorithms for the simple plant location problem. Computers and Operations Research. 31(2), 241–255.

Goldengorin, B., Jager, G., Molitor, P., 2006. Tolerances applied in combinatorial optimization. Journal of Computer Science. 2(9), 716–734.

Haimovich, M., Rinnooy Kan, A. H. G., Stougie, L., 1988. Analysis of heuristic routing problems. In: Golden et al., (eds): Vehicle routing: Methods and studies. Amsterdam : North Holland, 47–61.

Hansen, P., Mladenović, N., Moreno-Pérez, J.A., 2010. Variable neighbourhood search: methods and applications. Annals of Operations Research. 175(1), 367–407.

Jonker, R., Volgenant, A., 1986. Improving the hungarian assignment algorithm. Operations Research Letters. 5(4), 171–175.

Kara, I., 2008. Two indexed polonomyal size formulationsfor vehicle routing problems.Technical Report. Baskent University, Ankara/Turkey

Koltai, T., Terlaky, T., 2000. The difference between the managerial and mathematical interpretation of sensitivity analysis results in linear programming. International Journal of Production Economics. 65(3), 257–274.

Kuhn, H.W., 1955. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly, 2, 83–97.

Laporte, G., Nobert, Y., Desrochers, M., 1985. Optimal routing under capacity and distance restractions. Operations Research. 33(5), 1050–1073.

Laporte, G., Nobert, Y., 1987. Exact algorithms for the vehicle routing problem. Annals of Discrete Mathematics. 31, 147–184.

Laporte, G., Nobert, Y., Taillefer, S., 1987. A branch and bound algorithm for the asymmetrical distance-constrained vehicle routing problem. Mathematical Modelling. 9(12), 857–868.

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

Lawer, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., 1985. The traveling salesman problem : A guided tour of combinatorial optimization. Wiley-Interscience series in discrete mathematics. Chichester : Wiley.

Lenstra, J.K., Rinnooy Kan, A.H.G., 1975. Some simple applications of the traveling salesman problem. Operational Research Quaterly. 26(4), 717–734.

Lin, C., Wen, U., 2003. Sensitivity analysis of the optimal assignment. Discrete Optimization. 149(1), 35–46.

Mladenović, N., Hansen, P., 1997. Variable neighbourhood search. Computers and Operations Research. 24(11), 1097–1100.

Mladenović, N., Moreno-Prez, J.A., 2012. A survey of variable neighborhood search for solving vehicle routing problems. YUJOR. To appear.

Nemhauser, G.L., Wolsey. L.A., 1988. Integer and combinatorial optimization. Discrete mathematics and optimization. New York ; Chichester : Wiley.

Paschos, V.Th., 2009. An overview on polynomial approximation of NP-hard problems. Yugoslav Journal of Operations Research. 19(1), 3–40.

Pastor, R., Corominas, A., 2004. Branch and win: OR tree search algorithms for solving combinatorial optimisation problems. Top. 12(1), 169–191.

Pop, P., 2011. A new efficient transformation of the generalised vehicle routing problem into the classical vehicle routing problem, YUJOR, 21(2), 150–160.

Toth, P., Vigo, D., 2002. The vehicle routing problem. SIAM monographs on discrete mathematics and applications. Philadelphia, Pa. : Society for Industrial and Applied Mathematics.

Turkensteen, M., 2007. Source code: upper tolerance based algorithms for the `ATSP`. University of Groningen. World Wide Web http://www.informatik.uni-halle.de/ti/forschung/toleranzen/quelltexte/.

Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G., 2008. Tolerance-based Branch and Bound algorithms for the ATSP. European Journal of Operational Research. 189, 775–788.

Volgenant, A., 2006. An addendum on sensitivity analysis of the optimal assignment. European Journal of Operational Research. 169(1), 338–339.

Wolsey, L.A., 1998. Integer programming. Wiley-Interscience series in discrete mathematics and optimization. New York ; Chichester : John Wiley.