Tolerance-Based vs. Cost-Based Branching for the Asymmetric Capacitated Vehicle Routing Problem

Mikhail Batsyn, Boris Goldengorin, Anton Kocheturov, and Panos M. Pardalos

Abstract In this chapter, we consider the asymmetric capacitated vehicle routing problem (ACVRP). We compare the search tree size and computational time for the bottleneck tolerance-based and cost-based branching rules within a branch-and-bound algorithm on the FTV benchmark instances. Our computational experiments show that the tolerance-based branching rule reduces the search tree size by 45 times and the CPU time by 2.8 times in average.

Keywords Vehicle routing · Branch-and-bound · Tolerance · Branching strategy

1 Introduction

The vehicle routing problem (VRP) consists in finding K disjoint cyclic routes (one route for each of K vehicles) in a complete weighted graph starting and ending at the depot (vertex 0) and covering n clients (vertices $1, \ldots, n$) such that the total traveled distance (the total weight of all K routes) is minimized. In the capacitated vehicle routing problem (CVRP) every client i has a certain demand d_i of goods, which should be delivered from the single depot, and every vehicle has a limited capacity C of goods it can supply to clients. The CVRP is one of the difficult combinatorial

M. Batsyn (✉) · A. Kocheturov · P.M. Pardalos
Laboratory of Algorithms and Technologies for Networks Analysis, National Research University Higher School of Economics, 136 Rodionova, Nizhny Novgorod, Russian Federation
e-mail: mbatsyn@hse.ru

A. Kocheturov
e-mail: antrubler@gmail.com

P.M. Pardalos
e-mail: pardalos@ufl.edu

B. Goldengorin · P.M. Pardalos
Center of Applied Optimization, University of Florida, 401 Weil Hall, P.O. Box 116595, Gainesville, FL 32611-6595, USA

B. Goldengorin
e-mail: goldengorin@ufl.edu

B.I. Goldengorin et al. (eds.), *Models, Algorithms, and Technologies for Network Analysis*, Springer Proceedings in Mathematics & Statistics 59,
DOI 10.1007/978-1-4614-8588-9_1,

optimization problems (see, e.g., Toth and Vigo (2002) [15] and Golden et al. (2008) [9]). The largest instances which can be solved by the state-of-art algorithms have only 135 vertices (Lysgaard et al. (2004) [13], Baldacci et al. (2004) [1], Fukasawa et al. (2006) [6], Baldacci et al. (2008) [2], Baldacci et al. (2011) [3]).

The majority of the papers are devoted to the symmetric CVRP. Good overviews of the recent results for the CVRP could be found in papers of Laporte (2009) [12] and Baldacci et al. (2012) [4]. A complete and detailed description of different types of the VRP and algorithms for solving VRP problems is provided in the book of Toth and Vigo (2002) [15]. The asymmetric CVRP (ACVRP) has received less attention in recent decades. After the paper of Fischetti et al. (1994) [5], there are almost no exact algorithms for the ACVRP except maybe Pessoa et al. (2008) [14]. According to the results reported by Pessoa et al. (2008) [14], branch-and-bound algorithms are more successful in solving ACVRP problems with a small number of vehicles, while branch-and-cut-and-price approach is better for a greater number of vehicles. The best results for the ACVRP with 2–3 vehicles were obtained in Fischetti et al. (1994) [5].

In this chapter, we compare two branching rules within a branch-and-bound algorithm for the ACVRP. They are the tolerance-based branching rule, which uses the bottleneck upper tolerance and the classical cost-based branching rule. Our tolerance-based branching strategy is similar to the approach suggested by Goldengorin et al. (2004) [10] for the ATSP problem. Compared to the cost-based branching rule, our tolerance-based branching rule allows to reduce the search tree size considerably. The notion of tolerance is well known in sensitivity analysis for linear programming problems (see, e.g., Gal and Greenberg, 1997 [7]). A tolerance is a maximal possible change of a parameter value (an arc weight in the case of the ACVRP) for which the current optimal solution remains optimal. A tolerance-based approach was suggested by Goldengorin et al. (2004) [10] and proved its efficiency for the asymmetric traveling salesman problem (ATSP). In this paper, we present some preliminary computational results showing a comparison of the classical cost-based branching rule and the suggested tolerance-based one. Our computational experiments for seven FTV benchmark instances show that the search tree size is reduced by 45 times in average and the computational time is reduced by 2.8 times in average.

The chapter is organized as follows. In the next section, we give the formulation of the ACVRP. The suggested branch-and-bound algorithm is described in Sect. 3. The computational results are presented in Sect. 4. Section 5 concludes the chapter with a short summary.

2 The Asymmetric Capacitated Vehicle Routing Problem

The ACVRP has several mathematical programming models. Below we provide the two-index vehicle flow formulation (Toth and Vigo, 2002 [15]). The ACVRP is described by a directed weighted graph $G = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is a

set of vertices (vertex 0 is a depot, vertices $1, \ldots, n$ are clients) and A is a set of arcs. Arc weights are given by the cost matrix c_{ij}. There are K vehicles of identical capacity C located at the depot. A given quantity of goods d_i should be delivered to client i. The objective is to find K disjoint cyclic routes from the single depot, one route for each vehicle, serving all the n clients with their demands d_i such that the total traveling cost is minimized.

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \setminus \{0\} \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \setminus \{0\} \tag{3}$$

$$\sum_{i \in V} x_{i0} = K \tag{4}$$

$$\sum_{j \in V} x_{0j} = K \tag{5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq \left\lceil \sum_{i \in S} d_i / C \right\rceil \qquad \forall S \subseteq V \setminus \{0\}, S \neq \varnothing \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{7}$$

Variable x_{ij} is equal to 1 when arc (i, j) belongs to the optimal solution and is equal to 0 otherwise. The objective function (1) requires that the total weight of the arcs in the solution is minimized. Constraints (2) and (3) require that only one arc in the solution goes to a client and only one arc leaving this client. Constraints (4) and (5) require that there are K incoming and K outgoing arcs in the depot. Capacity-cut constraints (6) require that any set S of clients with total demand $\sum_{i \in S} d_i$ should be served by the number of vehicles with capacity C which is enough to cover such a demand.

3 Algorithm Description

In our algorithm, we use the assignment problem (AP) as a relaxation of the ACVRP. For this purpose, row c_{0j} and column c_{i0} of the cost matrix c_{ij} are duplicated K times. This means that we make K copies of the depot vertex in order to transform constraints (4) and (5) into standard AP constraints. The AP solution is represented by a set of disjoint cycles some of which could be infeasible for the ACVRP problem. There are two types of infeasible cycles: a cycle without the depot, and an

overloaded cycle with the total demand of clients greater than a vehicle capacity. In order to reduce the search tree size, we also consider as infeasible too small cycles with the total demand of clients smaller than $\sum_{i \in V} d_i - (K-1)C$. In the case of such a cycle, the remaining $(K-1)$ vehicles cannot serve the remaining clients because the total demand of these clients is greater than $(K-1)C$.

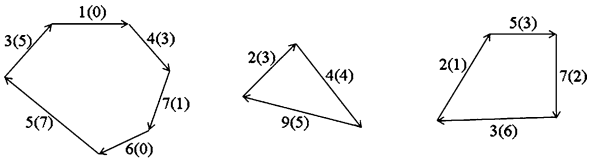
In each node of the solution tree, we solve the assignment problem. If the AP solution is feasible for the ACVRP, then the current branch is completed. Otherwise all the infeasible cycles are considered and one of them is destroyed by means of removing one of its arcs. To remove arc a, its traveling cost c_a is replaced with ∞. For example, if there is an infeasible cycle of three arcs x, y, z in the AP solution, then in the first branch of the solution tree we remove arc x (branch $\overline{x}$). In the second branch, this arc is forcibly included to the AP solution (by setting $c_x = -\infty$), and arc y is removed (branch $x\overline{y}$). Finally, in the third branch arcs x and y are forcibly included to the AP solution, and arc z is removed (branch $xy\overline{z}$). The question is which cycle is better to destroy and what arc to remove first.

We suggest to use upper tolerances for efficient selecting of a branch in a branch-and-bound algorithm. Upper tolerance u_a of arc a from the optimal AP solution is the maximal value on which the cost of this arc c_a can be increased so that this arc still belongs to the optimal solution. It is not difficult to prove that an upper tolerance can be computed using the following formula: $u_a = f_{-a} - f$, where f is the value of the AP objective function and f_{-a} is the value of this function for the AP in which arc a is removed (Goldengorin et al., 2004 [10]). Thus, upper tolerance u_a also shows how much the objective function is increased when we remove arc a.

We calculate upper tolerances of arcs in the infeasible cycles and first remove the arc which has the minimal tolerance in its cycle. This causes the minimal possible increase of the objective function (the total traveling cost), and thus we cannot miss the branch with the optimal solution of the original ACVRP problem. We call this minimal tolerance in a cycle a "cycle tolerance", because it shows the increase in the objective function which we cannot avoid when destroying this cycle. Since all the infeasible cycles cannot be present in the optimal solution of the original problem, then we cannot avoid the increase in the objective function equal to the maximal of the tolerances of infeasible cycles. We call this tolerance a "bottleneck tolerance"—the maximal tolerance among the minimal tolerances of arcs of each infeasible cycle. In order to find the optimal value of the original problem, it is necessary to destroy the cycle with the bottleneck tolerance because this tolerance gives the maximal increase in the objective function without jumping over its unknown optimal value for the ACVRP. A classical cost-based branching strategy selects the smallest cycle to destroy (with the smallest number of arcs) and first removes the longest arc (the arc with the greatest weight).

Let the three cycles shown in Fig. 1 be infeasible for some ACVRP. Numbers $c_a(u_a)$ near every arc show its weight c_a and upper tolerance u_a. The minimal tolerances of cycles are equal to 0, 3, 1, correspondingly. According to the tolerance-based branching strategy, the second cycle should be destroyed and its arc with weight 2 and tolerance 3 (which is a bottleneck tolerance) should be removed first. The cost-based approach also destroys the second cycle but removes the longest arc with weight 5 first.

Fig. 1 Infeasible cycles



In all the procedures of our algorithm described below, the following variables are used:

- V—the list of vertices in the graph
- A—the list of arcs in the graph
- x—the solution of the AP, $x_{ij} = 1$ if vertex j is assigned to follow vertex i in a cycle
- c—the cost matrix
- c_a—the cost (weight) of arc a
- c_a^0—the original cost of arc a
- u_a—the upper tolerance of arc a
- f—the total cost for the current AP solution
- f_{-a}—the total cost for the AP solution after removing arc a
- f^*—the total cost for the best feasible ACVRP solution found so far
- r—the current cyclic route (cycle) in the current AP solution
- R—the set of cyclic routes (cycles) representing the current AP solution
- $|r|$—the number of arcs in cycle r
- K—the number of vehicles
- C—the capacity of each vehicle
- d_i—the demand of client i
- d_r—the total demand of clients in the current cycle
- $d_{\min}$—the minimal total demand of a feasible cycle
- a^*—the arc with the bottleneck tolerance
- r^*—the cycle with the arc having the bottleneck tolerance
- u^*—the bottleneck tolerance
- $u_{\min}$—the minimal tolerance in the current cycle
- $a_{\min}$—the arc with the minimal tolerance in the current cycle

The main procedure of our algorithm is a recursive function BRANCH-AND-BOUND(c) (Algorithm 1). First, we solve a relaxed problem which is the assignment problem (AP) in our approach. We apply the JV algorithm by Jonker and Volgenant (1987) [11] for this purpose (function SOLVE-ASSIGNMENT-PROBLEM(c, R), Algorithm 2). A solution of the AP is a set of disjoint cycles covering all the vertices. The value f of the AP objective function gives a lower bound for the optimal value of the ACVRP objective function on the current branch of the search tree. That is why if this value is not less than the value f^* of the best feasible solution found so far, then this branch is completed.

Otherwise, we check if this solution is feasible for the ACVRP (function IS-FEASIBLE(R), Algorithm 3). If it is the case, this solution replaces the currently

Algorithm 1 Main recursive procedure

function BRANCH-AND-BOUND(c)
  f = SOLVE-ASSIGNMENT-PROBLEM(c, R)
  if $f > f^*$ **then**
    return
  end if
  if IS-FEASIBLE(R) **then**
    $f^* = f$
    return
  end if
  COMPUTE-TOLERANCES(c, f, R, a, r, u)
  while $f + u_a < f^*$ **do**
    $c_a^0 = c_a$ ▷ backup the original cost of arc a
    $c_a = +\infty$ ▷ remove arc a from the solution
    BRANCH-AND-BOUND(c)
    $c_a = -\infty$ ▷ include arc a to the solution
    $u_a = +\infty$ ▷ remove this tolerance from consideration
    $a = \arg\min_{a \in r}(u_a)$ ▷ find next minimal tolerance
  end while
  for $a \in \{a | c_a = -\infty\}$ **do** ▷ restore the original arc costs
    $c_a = c_a^0$
  end for
end function

Algorithm 2 Runs the JV algorithm to solve the AP

function SOLVE-ASSIGNMENT-PROBLEM(c, R)
  $f = \mathrm{JV}(c, x)$ ▷ JV algorithm by Jonker and Volgenant (1987) [11]
  Get cycles R from assignments x
  return f
end function

best solution and the branch is completed. To check the feasibility, we check every cycle r to have the depot (vertex 0) and the total demand of clients d_r not less than the minimal possible demand for one cycle $d_{\min}$ and not greater than the vehicle capacity C. The minimal cycle demand is calculated by the following formula: $d_{\min} = \sum_{i \in V} d_i - (K-1)C$. If the cycle total demand is less than $d_{\min}$ then the remaining $(K-1)$ vehicles capacity is not enough to cover the demand of the remaining clients.

If the AP solution has infeasible cycles, we calculate the upper tolerances u of arcs in these cycles and find the arc a^* with the bottleneck tolerance u^* and its cycle r^* (function COMPUTE-TOLERANCES(c, f, R, a^*, r^*, u), Algorithm 4). Note that among the cycles having the same maximal tolerance u^* we choose the one having the minimal size (number of arcs) $|r|$ because it provides less branches

Algorithm 3 Checks if cyclic routes in R are feasible, removes feasible cycles

```
function IS-FEASIBLE(R)
    for r ∈ R do                                  ▷ for each of the cyclic routes (cycles) R
        if 0 ∉ r then
            continue                              ▷ no depot in cycle r
        end if
        d_r = Σ_{i∈r} d_i                         ▷ total demand of clients in cycle r
        if d_r > C or d_r < d_min then
            continue                              ▷ overloaded or underloaded cycle
        end if
        R = R \ {r}                               ▷ remove feasible cycle r
    end for
    if R = ∅ then
        return true
    else
        return false
    end if
end function
```

in the search tree. Since calculation of tolerances (function COMPUTE-UPPER-TOLERANCE(c, f, a), Algorithm 5) is computationally difficult we stop the calculation of tolerances in non-bottleneck cycles as soon as possible. A cycle cannot have a bottleneck arc if it has an arc which tolerance is less than the currently maximal cycle tolerance u^* (or it is equal to u^* and the size of the cycle $|r|$ is not less than the size of the cycle $|r^*|$ having the currently maximal tolerance).

After the bottleneck cycle is found, we consequently remove one of its arcs in a while-loop and recursively call BRANCH-AND-BOUND(c) function for this branch. To remove an arc from the AP solution, we set its cost to $+\infty$ and to forcibly include an arc to the AP solution we set its cost to $-\infty$. Let the bottleneck cycle arcs $a_1, a_2, \ldots, a_m$ be ordered by its tolerances: $u_{a_1} \le u_{a_2} \le \cdots \le u_{a_m}$. Then we have the following m branches: $\overline{a_1}$, $a_1\overline{a_2}$, $a_1a_2\overline{a_3}$, $\ldots$, $a_1a_2\ldots a_{m-1}\overline{a_m}$. After all the branches are considered, we restore the original values of arc costs and return from recursion.

4 Computational Results

We perform our computational experiments on Intel Core i7 laptop machine with 1.9 GHz CPU and 8 Gb of memory. The search tree size (number of nodes) and computational time in seconds are shown in Table 1 for the classical cost-based branching rule and for the suggested tolerance-based one. In average, the search tree size is 45 times smaller for the suggested branching strategy. The computational time is 2.8 times smaller in average. Though for some instances, the running time

Algorithm 4 Finds upper tolerances u, bottleneck arc a^* and cycle r^*

```
function COMPUTE-TOLERANCES(c, f, R, a*, r*, u)
    u* = −1                                          ▷ bottleneck tolerance
    u_min = +∞                                       ▷ minimal tolerance in a cycle
    a_min = −1                                       ▷ the arc with the minimal tolerance in a cycle
    for r ∈ R do                                     ▷ for each cycle r of the infeasible cycles R
        for a ∈ r do                                 ▷ for each arc a in cycle r
            u_a = COMPUTE-UPPER-TOLERANCE(c, f, a)
            if (u_a < u*) or (u_a = u* and |r| ≥ |r*|) then   ▷ skip non-bottleneck cycle
                u_min = +∞
                break
            end if
            if u_a < u_min then
                u_min = u_a
                a_min = a
            end if
        end for
        if u_min = +∞ then
            continue                                 ▷ skip non-bottleneck cycle
        end if
        if (u_min > u*) or (u_min = u* and |r| < |r*|) then
            u* = u_min
            r* = r
            a* = a_min
        end if
    end for
end function
```

Algorithm 5 Computes the upper tolerance of arc a

```
function COMPUTE-UPPER-TOLERANCE(c, f, a)
    c_a^0 = c_a
    c_a = +∞
    f_−a = JV(c, x)
    c_a = c_a^0
    u_a = f_−a − f
    return u_a
end function
```

is slightly greater, for others it is considerably smaller. For example, it is 3 times smaller for FTV64 and 15 times smaller for FTV70 instances. The reduction in the search tree size varies from 2.8 times to 350 times.

Table 1 Search tree size and running time for cost-based and tolerance-based branching

$\|V\|$	Optimal value	Cost-based, nodes	Tolerance-based, nodes	Cost-based, sec	Tolerance-based, sec
33	1406	71555	5640	0.8	0.9
35	1644	137171	47492	2.2	5.4
38	1654	225737	29081	4.6	7.2
44	1740	1832873	88619	36.7	27.7
47	1891	151697	25863	3.5	7.8
55	1739	1324700	122381	41.2	52.1
64	1974	13981245	175153	523.2	173.5
70	2054	5534172	15797	194.1	12.2

5 Conclusion

The suggested tolerance-based branching strategy allows us to reduce the search tree size in a branch-and-bound algorithm considerably. In this chapter, we have presented preliminary results for a simple branch-and-bound algorithm without any efficient lower bounds and heuristic to find high-quality feasible ACVRP solutions. For example, one of the best exact algorithms for the ACVRP—the FTV algorithm by Fischetti et al. (1994) [5] computes the so-called disjunctive lower bound, then the lower bound based on min-cost flow relaxation, and finally applies the additive approach to get a tighter lower bound from these two bounds. The FTV algorithm also runs the VHVRP heuristic by Vigo (1996) [17] to obtain a feasible ACVRP solution. Our results for a full-fledged branch-and-bound algorithm with tight lower bounds and high-quality heuristic solutions will be published in a journal paper.

Since the suggested tolerance-based branch-and-bound algorithm requires solving of a great number of assignment problems (AP), it is reasonable to reduce the computational time spent for solving the AP. We apply one of the most efficient algorithms for the AP—the JV algorithm by Jonker and Volgenant (1987) [11]. Another idea is to use the algorithm of Volgenant (2006) [18] to compute an optimal AP solution and all its tolerances in $O(n^3)$ time. We can also try to reduce the quantity of solved assignment problems by selecting the smallest cycle and the arc with the minimal tolerance in it instead of looking for the bottleneck cycle (see Turkensteen et al. (2008) [16]). Another source of research is to check whether the lower tolerances will be helpful either to connect unserved cycles with the so called underloaded feasible cycles (see Germs et al. (2012) [8]).

Acknowledgements The authors are supported by LATNA Laboratory, National Research University Higher School of Economics (NRU HSE), Russian Federation government grant, ag. 11.G34.31.0057.

Mikhail Batsyn and Boris Goldengorin were supported by the project No11-04-0008: "Calculus of tolerances in combinatorial optimization: theory and algorithms", carried out within the Higher School of Economics 2011–2012 Academic Fund Program.

Mikhail Batsyn and Anton Kocheturov are supported by Federal Grant-in-Aid Program "Research and development on priority directions of development of the scientific-technological complex of Russia for 2007–2013" (Governmental Contract No. 14.514.11.4065).

References

1. Baldacci, R., Hadjiconstantinou, E., Mingozzi, A.: An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. Oper. Res. **52**, 723–738 (2004)
2. Baldacci, R., Christofides, N., Mingozzi, A.: An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Math. Program., Ser. A **115**(2), 351–385 (2008)
3. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Oper. Res. **59**(5), 1269–1283 (2011)
4. Baldacci, R., Mingozzi, A., Roberti, R.: Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. Eur. J. Oper. Res. **218**, 1–6 (2012)
5. Fischetti, M., Toth, P., Vigo, D.: A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper. Res. **42**, 846–859 (1994)
6. Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragao, M., Reis, M., Uchoa, E., Werneck, R.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Math. Program., Ser. A **106**, 491–511 (2006)
7. Gal, T.T., Greenberg, H.J. (eds.): Advances in Sensitivity Analysis and Parametric Programming. Internat. Ser. Oper. Res. Management Sci., vol. 6. Kluwer Academic, Boston (1997)
8. Germs, R., Goldengorin, B., Turkensteen, M.: Lower tolerance-based Branch and Bound algorithms for the ATSP. Comput. Oper. Res. **39**(2), 291–298 (2012)
9. Golden, B.L., Raghavan, S., Wasil, E.A.: The Vehicle Routing Problem: Latest Advances and New Challenges. Operations Research/Computer Science Interfaces, vol. 43. Springer, New York (2008)
10. Goldengorin, B., Sierksma, G., Turkensteen, M.: Tolerance based algorithms for the ATSP. In: Lecture Notes in Computer Science, vol. 3353, pp. 222–234 (2004)
11. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**, 325–340 (1987)
12. Laporte, G.: Fifty years of vehicle routing. Transp. Sci. **43**(4), 408–416 (2009)
13. Lysgaard, J., Letchford, A.N., Eglese, R.W.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Program., Ser. A **100**, 423–445 (2004)
14. Pessoa, A., de Aragao, M.P., Uchoa, E.: Robust branch-cut-and-price algorithms for vehicle routing problems. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges. Operations Research/Computer Science Interfaces, vol. 43, pp. 297–325 (2008)
15. Toth, P., Vigo, D.: The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia (2002)
16. Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G.: Tolerance-based branch and bound algorithms for the ATSP. Eur. J. Oper. Res. **189**(3), 775–788 (2008)
17. Vigo, D.: A heuristic algorithm for the asymmetric capacitated vehicle routing problem. Eur. J. Oper. Res. **89**(1), 108–126 (1996)
18. Volgenant, A.: An addendum on sensitivity analysis of the optimal assignment. Eur. J. Oper. Res. **169**(1), 338–339 (2006)