



GRASP Method for Vehicle Routing with Delivery Place Selection

Petar Afric^(✉), Adrian Satja Kurdija, Lucija Šikić, Marin Silic, Goran Delac, Klemo Vladimir, and Sinisa Srbljic

Faculty of Electrical Engineering and Computing, University of Zagreb,
Unska 3, Zagreb, Croatia

{petar.afric,adrian.kurdija,lucija.sikic,marin.silic,
goran.delac,klemo.vladimir,sinisa.srbljic}@fer.hr

Abstract. In this paper we present a greedy randomized adaptive search procedure (GRASP) for solving a vehicle routing problem (VRP) for package delivery with delivery place selection. The problem can be solved by stepwise optimization, i.e., first selecting delivery sites and then defining routes based on that selection. Alternatively, it can be solved by jointly optimizing delivery site selection and routing. We investigate the effects of stepwise optimization in comparison to joint optimization. The evaluation results show that our proposed stepwise approach, while expectedly producing longer routes than joint approach (by 4% on average), can provide a solution 1000× faster than the previous benchmark approach. The proposed procedure is therefore well suited for the dynamic environment of package delivery which is widespread in modern cities as a consequence of e-commerce.

Keywords: Greedy randomized adaptive search procedure · Vehicle routing problem · Package delivery

1 Introduction

Humanity is increasingly using e-commerce. In modern cities, especially smart cities of the future, there is an increasing need for optimized package delivery. The motivation is both to maximize profit and to reduce our impact on the environment. In this paper we study package delivery with delivery site selection.

Schittekat et al. present an interesting problem and analysis in [1]. They tackle a problem of school bus routing (SBR) with bus stop selection. In this problem, students need to be assigned to a station from a list of potential stations and then routes need to be constructed so that all students are transported to school while covering a distance as small as possible. By jointly optimizing student station assignment and station route assignment, they produce significantly shorter routes than if they divided the process into two steps: assigning

students to stations, and defining routes for the stations with assigned students. We noticed several things:

- This problem is equivalent to the problem of package delivery where the package can be delivered to multiple delivery sites. Since providing multiple delivery sites can produce shorter routes, such user behavior should be incentivized.
- The running time of the joint optimization approach seems quite large for the dynamic world of package delivery. We decided to investigate how time performance improves when using a stepwise approach.
- The benchmark stepwise approach presented in the above paper seems very simple, thus artificially increasing the benefit of the presented approach. We decided to investigate how a smarter stepwise approach compares to the joint approach.

The rest of the paper is organized as follows. Section 2 defines the problem in precise terms. Section 3 presents the work related to this paper. In Sect. 4, a detailed description of the proposed algorithm is given. Section 5 presents the evaluation results. Section 6 gives final comments about our work and future research possibilities.

2 Problem Definition

In this paper the problem of defining vehicle routes for delivery trucks with an addition of delivery place selection is considered. The problem can be described as follows:

- there is a single production factory,
- there are N delivery stations,
- there are M customers who will pick up their delivery at the delivery station.

Furthermore,

- each customer will visit only one delivery station, which needs to be in his or her walking range,
- each customer will pick up only one delivery package,
- all delivery trucks have the same capacity which indicates the number of packages they can carry (all packages are assumed to have the same size),
- any delivery station can be visited by only one delivery truck.

The goal is to minimize the total route length travelled by all trucks.

A mathematical description of the problem is shown in Table 1 and Eqs. 1, 2, 3, 4 and 5.

The aim is to minimize expression (1):

$$\sum_{i=1}^N \sum_{j=1}^N c_{ij} \sum_{k=1}^T x_{ijk} + \sum_{i=1}^N cp_i \sum_{k=1}^T xp_{ik} \quad (1)$$

Table 1. Mathematical problem description

Parameter	Description
N	Number of delivery stations
M	Number of customers
T	Number of delivery trucks
K_k	Capacity of delivery truck k specified by number of packages it can carry
c_{ij}	Cost of going from station i to station j
cp_i	Cost of going from station i to the production factory
pn_l	Number of packages picked up by customer l
s_{il}	1 if the station i is within range of customer l , otherwise 0
x_{ijk}	1 if delivery truck k travels between stations i and j , otherwise 0
xp_{ik}	1 if delivery truck k travels between stations i and the production factory, otherwise 0
y_{ik}	1 if the delivery truck k goes to station i , otherwise 0
z_{il}	1 if the customer l goes to station i , otherwise 0

While respecting the following restrictions:

$$\sum_{i=1}^N z_{il}s_{il} = 1 \quad \forall l \in \{1, \dots, M\} \quad (2)$$

$$pn_l = 1 \quad \forall l \in \{1, \dots, M\} \quad (3)$$

$$K_i = K_j \quad \forall i, j \in \{1, \dots, T\} \quad (4)$$

$$\sum_{k=1}^T y_{ik} = 1 \quad \forall i \in \{1, \dots, N\} \quad (5)$$

In this paper we present a fast greedy randomized adaptive search procedure (GRASP) heuristic algorithm for solving the described problem. The algorithm consists of two parts:

1. assigning customers to delivery stations,
2. defining routes for visiting delivery stations.

The first part is done using a greedy heuristic, while the second one is done using a specialized local search. These two steps are repeated R times and the best solution is kept.

In order to evaluate the proposed approach, we conducted extensive laboratory experiments. During the experiments we compared various approach-represented in [1]. Our results show that although a stepwise approach does, on average, produce longer routes the difference gap can be minimized from 23% to 4% while improving execution time by three orders of magnitude. In the end we give guidelines for future work on this topic.

3 Related Work

Vehicle routing is a very active field of research. An overview of research in this field is given by Vigo and Toth in [2]. A large body of work in this field is concerned with incorporating as many real world constraint as possible. This is easily seen by the amount of variation of the vehicle routing problem (VRP). There is VRP with time windows (VRPTW), inventory routing problem (IRP), production routing problem (PRP), location routing problem (LRP), inventory location routing problem (ILRP), capacitated vehicle routing problem (CVRP), split delivery routing problem (SDVRP) and many others, all of them described in [3].

However, to the best of our knowledge there does not seem to be a large body of work which studies the effect of multiple delivery site availability. Most of the work on this topic seems to be done by dealing with the school bus routing (SBR) problem since station selection is a common part of this problem. [4] describe two stepwise approaches for such problems. Location-Allocation-Routing (LAR) first selects the delivery sites and then performs routing on the selected sites. Allocation-Routing-Location (ARL) first performs routing and then performs delivery site selection.

In our paper we have decided to use the LAR strategy and implement it using a GRASP metaheuristic [5], noting that Park and Kim observed in [6] that only a few metaheuristic approaches have been tried for this problem.

4 Proposed Algorithm

As introduced before, the algorithm can be divided into two parts:

1. assigning customers to delivery stations,
2. defining routes to delivery stations.

The following sections describe each of the steps.

4.1 Assigning Customers to Delivery Stations

The main idea behind the assignment algorithm is to reduce the number of delivery stations which need to be visited while preferring delivery stations which are closer to the production factory. The algorithm (shown formally in Algorithm 1) goes as follows:

- The fitness of each station is calculated. The fitness indicates how fit a station is to be assigned to customers. It is generated in the following way:
 - For each station it is initialized to zero.
 - A constant C (controlling the relevance of the distance to the factory) is randomly generated uniformly from $[0, 10]$. For each station, the fitness is increased by C divided by the distance of the station to the production factory.

- Then, for each customer, the fitness of each reachable station is increased by a reciprocal of the number of reachable stations. In this way the fitness of stations which can be reached by a lot of customers or which are the only option for some customers are increased.
- The stations are then sorted in descending order by fitness. If the fitness difference is less than $fitness_d_min$, precedence is given to stations closer to the production factory. $fitness_d_min$ is a hyper-parameter which we usually set to 0.01.
- The stations are then iterated in the sorted order. For each station, customers which can reach it are considered.
 - If the station can be reached by a number of customers less or equal to the capacity of the truck, then all the customers are assigned to that station.
 - If more customers can reach the station, they are sorted by the number of stations they can reach in ascending order. If two customers can reach the same number of stations, precedence is given to the customer which is farther from the production factory. Customers are then assigned to the current station in the sorted order until the capacity on the truck is filled.

After all of this is done, all customers are assigned to a station. Those stations which have at least one customer assigned to them are active stations and only they are considered in the rest of the algorithm.

Algorithm 1. Assigning customers to delivery stations

```

C ← getRandomValue(0, 10)
for station ← stations do
  fitness[station] = C/distance[factory][station]
end for
for customer ← customers do
  fconst = 1/stationsInCustomerReach[customer].Count()
  for station ← stationsInCustomerReach[customer] do
    fitness[station] += const
  end for
end for
Sort(stations).Ascending().Using(fitness, distance)
for station ← stations do
  customersInReach = customerInReach[station]
  if customersInReach ≤ truckCapacity then
    for customer ← customersInReach do
      stationFor[customer] = station
    end for
  else
    Sort(customersInReach).Descending()
    .Using(stationsInCustomerReach, distance)
    for i ← Range(truckCapacity) do
      customer = customersInReach[i]
      stationFor[customer] = station
    end for
  end if
end for

```

4.2 Defining Routes to Delivery Stations

The goal of this step is to define truck routes for active stations which are as short as possible while respecting the truck capacity constraint. This problem can be divided into two subproblems:

- determining which stations are grouped to the same route,
- determining the best possible order of visiting the stations in the route.

The developed algorithm can be divided into three separate steps:

1. preprocessing,
2. initial solution generation,
3. solution optimization.

Preprocessing involves iterating over all stations and defining routes for stations which cannot be combined with any other station due to the capacity constraint. These stations and routes are no longer taken into consideration in further optimization.

Initial solution generation is done using a greedy heuristic which does the following:

1. it selects a station which is not yet assigned to any route and creates a route for it,
2. then it iterates over the rest of the unassigned stations and calculates which of the *satisfiable* stations is closest to the set of stations currently in the route. A station is *satisfiable* if the distance between the station and the route is smaller than between the station and the factory.
3. If the closest station is found, it is added to the current route; otherwise that is the end of the current route creation.
4. The previous steps are repeated as long as there are active unassigned stations.

The described algorithm is shown in Algorithm 2. Each of the defined routes is optimized using a greedy heuristic TSP (travelling salesman problem) solver which will be described later.

Solution optimization is done using local search. The search is defined by the following properties.

- An incomplete neighborhood consists of one element which is generated by doing one of the following:
 - switching a station from one route to another,
 - selecting two stations from different routes and swapping the stations between the two routes,
 - joining two routes into one,
 - breaking a route into two routes at a random point.

Algorithm 2. Initial solution generation

```

route = {}
while activeStations.IsEmpty() == false do
  station = activeStations.RemoveAt(0)
  route.Add(station)
  routeExpanded = true
  while routeExpanded do
    routeExpanded = false
    closestToRoute = NULL
    for activeStation ← activeStations do
      distanceToRoute = Infinity
      for routeStation ← route do
        if distance[factory][activeStation] > distance[activeStation][routeStation] then
          distanceToRoute =
            Min(distanceToRoute, distance[activeStation][routeStation])
        end if
      end for
      if closestToRoute.distanceToRoute > distanceToRoute then
        closestToRoute = activeStation
        closestToRoute.distanceToRoute = distanceToRoute
      end if
    end for
    if closestToRoute! = NULL then
      route.Add(closestToRoute)
      routeExpanded = true
    end if
  end while
  tspSolver.Optimize(route).Using(distance)
  routes.Add(route)
  route = {}
end while

```

There is a small probability (a parameter called *repetitionChance*) that another modification occurs after each modification.

- Routes are re-optimized using a TSP solver in each iteration as soon as they are manipulated.
- The objective function is the sum of lengths of all routes.
- The stopping condition is reaching a maximum number of iterations or a maximum number of stagnant iterations. These are defined using hyperparameters *maxIterationCount* and *maxStagnationCount*.

The defined algorithm is shown in Algorithm 3.

Algorithm 3. Solution optimization

```

maxIterationCount = 1000000
maxStagnationCount = 100000
bestDistance = CalculateDistance(routes, distance)
while ++ generationNumber < maxIterationCount AND stagnation <
  maxStagnationCount do
    neighbor = Manipulate(routes, distance)
    neighbourDistance = CalculateDistance(neighbor, distance)
    if neighbourDistance < bestDistance then
      routes = neighbour
      bestDistance = neighbourDistance
      stagnation = 0
    else
      stagnation ++
    end if
  end while

```

The result produced by this step are the final routes. We are left to describe the TSP solver used in our problem.

A *TSP solver* is used to optimize the generated routes. This TSP solver uses the following greedy heuristic.

- If it is given three or less nodes, a list of the given nodes is returned as the solution.
- Otherwise, three random nodes are taken and declared the current optimal route. Then the following is iteratively done:
 - select a random node,
 - iterate over each edge of the current optimal route and calculate the distance change if that edge is removed and two new ones are added connecting the previously randomly selected node,
 - remove the edge which produced the minimal change in distance.
 This is repeated as long as there are nodes which are not in the route.
- Finally it returns the created route.

The described algorithm is shown in Algorithm 4.

Algorithm 4. TSP solver procedure

```

if nodes.Count() ≤ 3 then
  RETnodes
end if
route = {nodes.RemoveAt(random),
nodes.RemoveAt(random), nodes.RemoveAt(random)}
while nodes.IsEmpty() == false do
  node = nodes.RemoveAt(random)
  bestEdgeChange = Infinity
  bestEdge = NULL
  for edge ∈ Edges(route) do
    edgeChange = CalculateChange(route, edge, node)
    if edgeChange < bestEdgeChange then
      bestEdge = edge
      bestEdgeChange = edgeChange
    end if
  end for
  route.Add(node).ByBreaking(bestEdge)
end while

```

5 Results

For our experiments, we use the dataset presented in [1] and give our results relative to the results presented in that paper. The dataset contains 112 problem instances, with the number of customers ranging from 25 to 800, the number of stations ranging from 5 to 80, and the maximum allowed walking distance ranging from 5 to 40. For each value of the repetition parameter, we solve each instance 100 times and take the average solution route length and calculation time as the results for that instance.

Figure 1 shows the average solution route length increase produced by our algorithm when compared to the results of the joint presented in [1]. At a 100 repetitions our algorithm produces, on average, 4% longer routes. This is clearly superior compared to the 23% longer routes produced by the stepwise method reported in [1].

Figure 2 shows the solution route length increase in relation to the instance maximum walking distance constraining. As the maximum walking distance varies from 5 to 40, the increase in the route length for our solutions varies from -0.54% to 4.5% at 100 repetitions. Our results again outperform results in [1] which reports that, when varying the maximum walking distance from 5 to 40, the route length increases from 1.4% to 63.4%.

In addition, Figs. 3 and 4 present the relation between station/customer count and route length increase. As the amount of stations varies from 5 to 80, the increase in the route length for our solutions varies from 0% to 8.31% at 100 repetitions. As the amount of customers varies from 25 to 800, the increase in the route length for our solutions varies from 0% to 6.98% at 100 repetitions.

The time performance results depict the logarithm of time decrease in order to improve readability. Figure 5 shows the time decrease of our algorithm when compared to [1]. Our algorithm (on average) takes $e^{7.02} \approx 1100\times$ less time at 100 repetitions, which is clearly a superior result. The time decrease is due to the problem simplification which occurs as a consequence of the stepwise approach.

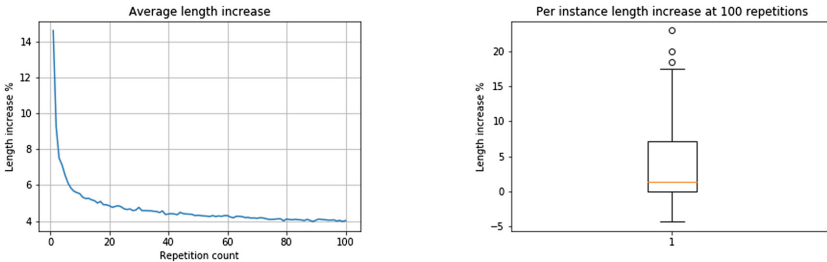


Fig. 1. Average solution route length increase

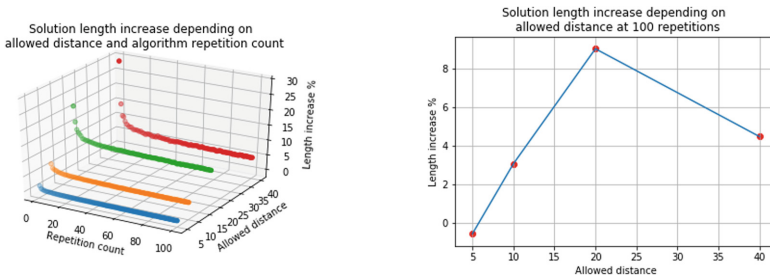


Fig. 2. Solution quality in relation to walking distance

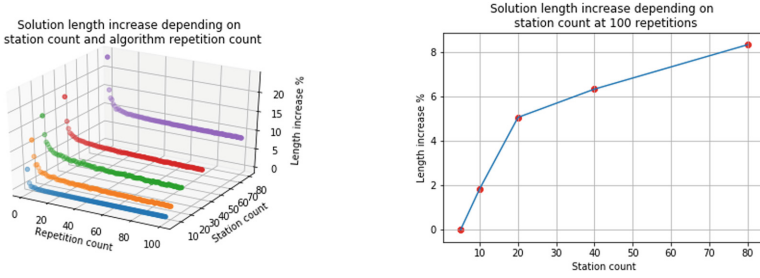


Fig. 3. Solution quality in relation to station count

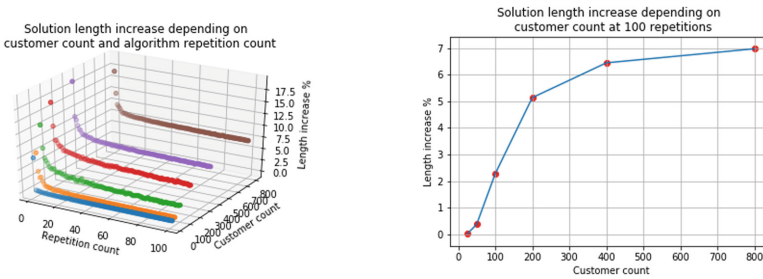


Fig. 4. Solution quality in relation to customer count

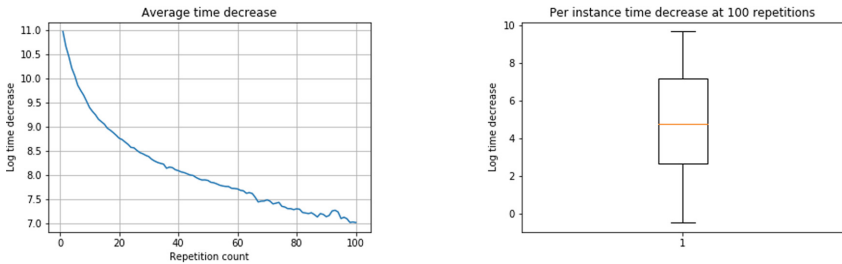


Fig. 5. Average time decrease

Figures 6, 7 and 8, show how the running time depends on the maximum walking distance, the number of stations, and the number of customers. As the maximum walking distance varies from 5 to 40, the time decrease for our algorithm varies from $e^{7.07} \approx 1171.13 \times$ to $e^{6.82} \approx 913.75 \times$ at 100 repetitions. As the number of stations varies from 5 to 80, the time decrease for our algorithm varies from $e^{2.01} \approx 7.47 \times$ to $e^{8.53} \approx 5036 \times$ at 100 repetitions. As the number of customers varies from 25 to 800, the time decrease for our algorithm varies from $e^{5.11} \approx 165 \times$ to $e^{7.8} \approx 2436 \times$ at 100 repetitions.

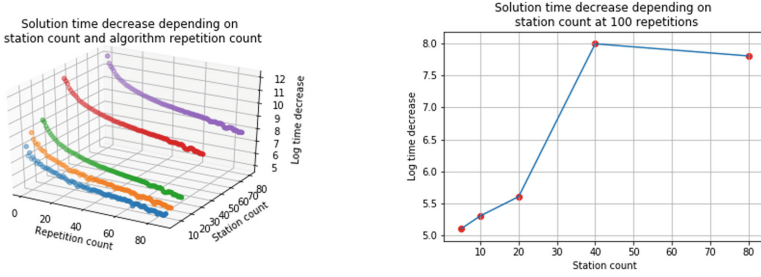


Fig. 6. Solution time decrease in relation to station count

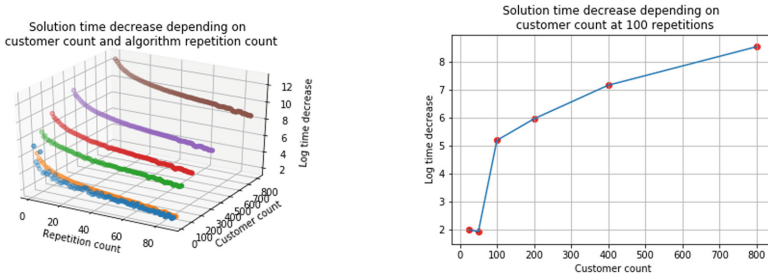


Fig. 7. Solution time decrease in relation to customer count

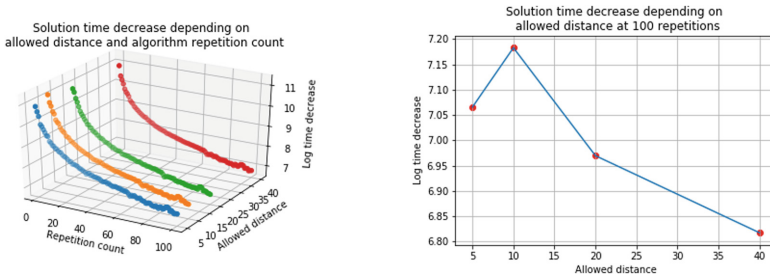


Fig. 8. Solution time decrease in relation to maximum walking distance

6 Conclusion

In this paper, a fast GRASP-based heuristic algorithm for vehicle routing with delivery place selection is presented. The evaluation results have shown that the proposed algorithm is, on average, three orders of magnitude faster than benchmark, thus showing that stepwise optimization can be much faster than joint optimization for this problem. We have also shown that using a smart stepwise approach can significantly reduce the performance gap with respect to the joint approach. Namely, with the proposed algorithm the gap has been reduced from 23% to 4%. In future work, our aim will be to increase the solution quality while maintaining or only slightly sacrificing the speed of the algorithm.

Acknowledgment. This research has been partly supported by the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS).

The authors acknowledge the support of the **Croatian Science Foundation** through the *Reliable Composite Applications Based on Web Services (IP-01-2018-6423)* research project.

The Titan X Pascal used for this research was donated by the NVIDIA Corporation.

References

1. Schittekat, P., Kinable, J., Sörensen, K., Sevaux, M., Spiessma, F., Springael, J.: A metaheuristic for the school bus routing problem with bus stop selection. *Eur. J. Oper. Res.* **229**(2), 518–528 (2013)
2. Toth, P., Vigo, D.: An overview of vehicle routing problems, pp. 1–26 (2001)
3. Archetti, C., Speranza, M.G.: A survey on mathheuristics for routing problems. *EURO J. Comput. Optim.* **2**(4), 223–246 (2014)
4. Laporte, G., Nobert, Y., Taillefer, S.: Solving a family of multi-depot vehicle routing and location-routing problems. *Transp. Sci.* **22**(3), 161–172 (1988)
5. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. *J. Glob. Optim.* **6**(2), 109–133 (1995). <https://doi.org/10.1007/BF01096763>
6. Park, J., Kim, B.I.: The school bus routing problem: a review. *Eur. J. Oper. Res.* **202**(2), 311–319 (2010)