The Capacitated Vehicle Routing Problem

Tomáš Turek

October 7, 2024

1 Definitions and tasks

Firstly we will repeat what is defined and given by the task and perhaps also add some of our own observations. *Note that the original statement of the task can be found here.*

1.1 The Capacitated Vehicle Routing Problem (CVRP)

We are **given**:

- A *depot*,
- A fleet of identical *vehicles* with capacity Q,
- n customers with *demands* $q_i \leq Q$, for $i = 1, \ldots, n$, to be delivered from the depot,
- Symmetric *travel cost* $c_{i,j}$ between points i and j for all pairs of points (i.e., the customers and the depot).

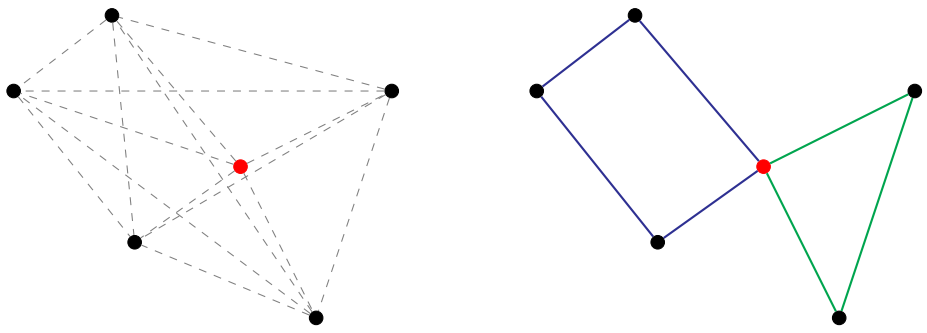
The **goal** is to determine routes of minimum total travel cost, subject to the following constraints:

- Each customer is serviced exactly once and she gets the desired demand,
- The total quantity delivered on each route does not exceed Q,
- Each route begins and ends at the depot.

1.1.1 Easy observations

Firstly we see that since all vertices must be accessible from all other ones we obtain a complete graph. For example see Figure 1a.

Next observation is that if we would use all n vehicles it would be simply from depot to one customer for each vehicle. Thus we want to create such routes which typically visits more customers. See Figure 1b for an example of the routes.



(a) Underlying structure of the graph. (b) Two possible routes.

Figure 1: Simple example of CVRP instance.

1.2 Tasks

Our task is to:

- To describe an integer linear programming (ILP) formulation of the problem,
- For each instance of the problem specified below, using your favorite software package for mathematical programming,
  - find an optimal solution of the LP relaxation derived from your ILP,
  - find an integral solution as good as possible (if doable),
  - provide the best known upper and lower bounds on optimal solution,
- Write a brief report.

2 Integer linear program

Lets firstly start by describing integer linear program that would solve our problem.

2.1 Variables

Let us define a variable x_{ij} for $i \neq j \in V$. This will be define in the following way. From the definition we have that all customers will be serviced exactly once, thus the edge will be used at most once.

$$x_{i,j} = \begin{cases} 1 & \text{if a truck goes through the edge } ij \\ 0 & \text{otherwise} \end{cases}$$

Next we need to measure the amount of goods that is being carried by each truck. For such case we will define $y_{i,j}$ for $i \neq j \in V$ which exactly states how much good is carried over the edge ij. This will be in between 0 and the total capacity Q.

Lastly the number of trucks is also unknown and thus we might add variable k which will be somewhere inside the boundaries $[0, n]$.

2.2 Optimization

From the stated variables we can easily describe the optimization function which is

$$\min \sum_{i \in V} \sum_{j \in V, j \neq i} c_{ij} x_{ij}.$$

Because we want to minimize how much travel we make by our trucks.

2.3 Constraints

Now we have to establish constraints. Clearly we have to ensure that the number of trucks entering to a customer and leaving the customer is both 1 (alternatively we could just write that they have to be equal, but only one truck will service the customer). And there will be exactly k trucks entering and leaving depot. We will be denoting depot by d.

$$\begin{aligned} \forall i \in V \setminus \{d\} : \sum_{j \in V, j \neq i} x_{ij} &= 1 \\ \forall i \in V \setminus \{d\} : \sum_{j \in V, j \neq i} x_{ji} &= 1 \\ \sum_{j \in V, j \neq d} x_{dj} &= k \\ \sum_{j \in V, j \neq d} x_{jd} &= k \end{aligned}$$

Next we have to ensure that the trucks will indeed service all customer without violating the capacities. So we need to sum all ingoing values and subtract the customer demand.

$$\forall i \in V \setminus \{d\} : \sum_{j \in V, j \neq i} y_{ji} - \sum_{j \in V, j \neq i} y_{ij} = q_i$$

Some of us may already know that one common problem is that closed circuits without the depot can arise in the solution. Luckily this cannot happen in our case since the very last constraint discards such options.

Lastly we have to establish the connection between x and y variables. This can be done simply by introducing another constraint.

$$\forall i \neq j \in V : y_{i,j} \leq Q \cdot x_{i,j}$$

2.4 ILP

Now we have established all parts of the integer linear problem and we will only recap and rewrite it in one place. And also we will show the bounds for each variable.

$$\begin{aligned}
& \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V, j \neq i} x_{ij} = 1 \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V, j \neq i} x_{ji} = 1 \\
& \sum_{j \in V, j \neq d} x_{dj} = k \\
& \sum_{j \in V, j \neq d} x_{jd} = k \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V, j \neq i} (y_{ji} - y_{ij}) = q_i \\
\forall i \neq j \in V : & \; y_{i,j} \leq Q \cdot x_{i,j} \\
\forall i, j \in V : & \; x_{ij} \in \{0, 1\} \\
\forall i, j \in V : & \; 0 \leq y_{ij} \leq Q \\
& \; 0 < k \leq n
\end{aligned} \tag{1}$$

3 Linear relaxation

Now lets talk for a while about a linear relaxation of this integer program 1. One can already see that it will behave more or less like a system of flows in the complete graph which will satisfy all constraints.

Next we may ask ourselves how to used such result to create a solution for given problem. One of the commonly used methods is to use the linear program result as a probability distribution. Which will be indeed our case. Therefore the linear program can be seen here 2.

$$\begin{aligned}
& \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V} x_{ij} = 1 \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V} x_{ji} = 1 \\
& \sum_{j \in V} x_{dj} = k \\
& \sum_{j \in V} x_{jd} = k \\
\forall i \in V \setminus \{d\} : & \sum_{j \in V, j \neq i} (y_{ji} - y_{ij}) = d_i \\
\forall i \neq j \in V : & \; y_{i,j} \leq Q \cdot x_{i,j} \\
\forall i, j \in V : & \; 0 \leq x_{ij} \leq 1 \\
\forall i, j \in V : & \; 0 \leq y_{ij} \leq Q \\
& \; 0 < k \leq n
\end{aligned} \tag{2}$$

3.1 Approximation

Lets describe our approximation algorithm. Pseudocode can be seen as Algorithm 1, but we will briefly describe it also in words. We will traverse the graph using x values to determine next step and control both the capacity and which customers were already visited. When we arrive in depot once again we may reset the capacity.

Algorithm 1 Approximation

Require: x, y from LP 2 and instance of the problem.
Ensure: Integer solution.

```
current = d
capacity = Q
while There exist not visited customer. do
    Filter neighbors based on x, capacity and if they were visited or not.
    if No such neighbors exists. then
        Choose them without considering x. And give them equal chances.
    end if
    Choose one of the neighbors based on their x values. Set it as current and decrease capacity.
    Update integer result.
    if We ended in depot. then
        Reset capacity to Q.
    end if
end while
```

Another common practice is to run such algorithm more times and choose the best result. Which is also something we will be using in our implementation.

4 Instances and solutions

We will firstly describe used tools and details of programs that we developed.

4.1 Solver

For the exact solutions we used Gurobi solver that is presented by the NEOS Server. To be exact we used the LP formulation for the server. Which also implies that the parameters of used computer won't be presented.

Hence we implemented a program written in Rust. We will shortly write what does the program do. When running the program we need to give some arguments to it. Firstly we have to choose if we want to generate ILP, LP or create approximation. For these we have options `ilp`, `lp` and `apx` respectively. Next we need to give the file path to the instance file and for approximation we also need to write the file path to the solution file. Last argument `-r` is for cargo only to use release version, therefore faster.

```
cargo run ilp path/to/vrp_file -r
cargo run lp path/to/vrp_file -r
cargo run apx path/to/vrp_file path/to/sol_file -r
```

We also present `checker.py` as a small python script which can take ILP solutions and write down trucks and their capacities.

Lastly we also added `parser.sh` for simplification of the parsing process and similarly `apx.sh` for approximating all results.

All of this will be given in an attachment. So it can be rerun, well except using NEOS Server. Therefore the solutions will be added as well.

Usually when having randomized algorithm we need to give a certain seed for the reproducibility. But in our case we skipped that, so the output of `apx.sh` will be also added. The reason for this is that we run the algorithm multiple times and the result are always pretty similar.

4.2 Exact solutions

After running the first seven instances with integer bounds we were able to get the result in somewhat reasonable time, therefore we can present such results for both integer and linear program.

For the next instances we could only use the linear relaxation and then use our simple approximation algorithm which was described earlier.

Also the known upper and lower bounds are presented within the input as a comment. To be exact usually the optimal solution is given which is indeed both upper and lower bound. In the very last two instances are only best solutions known which are the upper bounds.

Instance	**LP relaxation**	**ILP solution**	**Approximation**	**Upper bound**	**Lower bound**
E-n13-k4	217	247	294	247	247
E-n22-k4	307	375	441	375	375
E-n23-k3	471	569	629	569	569
E-n30-k3	418	505	532	534	534
E-n31-k7	330	379	461	379	379
E-n33-k4	717	838	1064	835	835
E-n51-k5	474	525	703	521	521
E-n76-k7	598	–	997	682	682
E-n76-k8	640	–	1082	735	735
E-n76-k10	714	–	1231	830	830
E-n76-k14	858	–	1501	1021	1021
E-n101-k8	736	–	1307	817	–
E-n101-k14	939	–	1698	1071	–

Table 1: Results using our solution and known bounds.

5 Conclusion

In the table 1 we may see that the LP relaxations are quite similar to the optimal known solutions. Also note that the lower bound for the last two instances can be taken from our solution.

Before we move on exact results lets talk about test **E-n30-k3**. For this one we got better result in integer program that is due to the fact of using more vehicles. Which in exact statement of this problem is also included the constraint that the smallest number of vehicles must be used. This can be obtained by adding a new element to the sum of the minimization function.

$$k \cdot \sum_{i<j} c_{i,j}$$

Next if see our integer program solutions we may get the same results. The sixth and seventh instance differ a little, which can be addressed to the rounding errors.

Lastly we can also see the results of our approximations. We may recall that the algorithm itself was pretty simple and now if we take a look at the results we are getting values which are not so far from the best known bounds. Also we may see that the approximations are not higher than twice the optimum. So from this we could estimate it is 2-approximation. Yet we cannot claim that it holds.