Università di Pisa

The Short-haul Distribution Problem

Model-Driven Decision Making Methods Project 2021/22

Wednesday 18th January, 2023

Bruno Javier Limón Avila - 646137
b.limonavila@studenti.unipi.it

Contents

1 Description of the problem

A logistic company has to organize the short-haul distribution of goods from its central depot to a set of customers in the same city/region; this can be represented by a directed graph with a node for the depot and one for each customer, and two directed arcs for each pair of nodes, in both directions, labeled with the distance between the two (which can be different between the two directions due to one-way streets). The company has a given supply of vehicles stationed at the deposit, of two different sizes: the small-size one carrying half of the total maximum amount of goods as the large-size one. However, it is assumed that goods are "flexible" enough so that they can always be arranged in the vehicle provided that their total amount does not exceed the capacity. Each customer requires a certain amount of goods, and has to be serviced by at most one vehicle of each type (and, clearly, at least by one vehicle). The problem is to serve all the customer at minimal total cost, which is (proportional to) the total distance traveled by all the vehicles and to a factor that depends on the type of the vehicle; in particular, the per-km cost of small-size ones is 75% of the cost of large-size ones.

As seen from the description, we are dealing with a variant of the classic Capacitated Vehicle Routing Problem CVRP, with the addition of Asymmetry due to the cost matrix being asymmetrical since the distances of the two directed arcs for each pair of nodes can be different, i.e. $c_{ij} \neq c_{ji}$. Besides this, one also needs to take into account the fact that the fleet of vehicles is Heterogeneous, having two different kinds of vehicles with different capacities and costs. Thus we end up with an AHCVRP problem.

1.1 Formulating an MILP model for the distribution problem.

1.1.1 Defining the input data and decision variables

- let $G = (V, A)$ be a complete and directed graph such that $V = \{0, 1, ...n\}$ is the set of nodes, where 0 denotes the central depot and $N = \{1, ...n\}$ is the set of customers with deterministic demand $d_i \geq 0$, node V_0 is the depot and has zero demand, $d_0 = 0$. A denotes the arcs between nodes
- T = {1,2} the types of available vehicles, where 1 is the large vehicle and 2 the small one
- K = {1,...,m} the number of vehicles available to use
- let $K1, K2$ be subsets of K, where $K1$ = {1,...,S_1}, representing the large vehicles and $K2 = K \backslash \{K1\}$ representing the small ones
- d_i = amount of goods required from customer i, $\forall i \in V \backslash \{0\}$
- S_t = number of t type of vehicles stationed at the depot, $\forall t \in T$
- c_{ij}^t = cost matrix taking into account distances between (i, j) and per-km cost factor for each t vehicle, $\forall (i, j) \in A$, $\forall t \in T$
- Q_k = maximum capacity of k vehicle, $\forall k \in K$
- $x_{ijk} = \begin{cases} 1 & \text{if arc } i,j \text{ belongs to the tour of } k \text{ vehicle} \\ 0 & \text{otherwise} \end{cases}$, $\forall (i, j) \in A$, $\forall k \in K$
- $z_{ik} = \begin{cases} 1 & \text{if customer } i \text{ is served by } k \text{ vehicle} \\ 0 & \text{otherwise} \end{cases}$, $\forall i \in N$, $\forall k \in K$
- y_{ik} = amount of goods delivered to customer i by vehicle k, $\forall (i) \in N$, $\forall (k) \in K$
- u_{ik} = support variable for the Miler-Tucker-Zemlin constraints $\forall (i) \in N$, $\forall (k) \in K$

1.1.2 Proposed MILP Model

Minimize

$$\sum_{(i,j)\in A}\sum_{k\in K1} c_{ij}^{1} x_{ijk} + \sum_{(i,j)\in A}\sum_{k\in K2} c_{ij}^{2} x_{ijk}$$

Subject to

$$\sum_{i\in V\setminus\{h\}} x_{ihk} - \sum_{j\in V\setminus\{h\}} x_{hjk} = 0, \forall h \in V \setminus \{0\},\ \forall k \in\ K \tag{1}$$

$$\sum_{j\in V\setminus\{0\}} x_{0jk} = 1, \forall k \in K \tag{2}$$

$$\sum_{k=1}^{K} y_{ik} = d_i, \forall i \in V \setminus \{0\} \tag{3}$$

$$\sum_{i\in V\setminus\{0\}} y_{ik} \leq Q_k, \forall k \in K \tag{4}$$

$$d_i z_{ik} \geq y_{ik}, \forall i \in V \setminus \{0\},\ \forall k \in K \tag{5}$$

$$\sum_{j\in V\setminus\{i\}} x_{ijk} = z_{ik}, \forall i \in V \setminus \{0\},\ \forall k \in K \tag{6}$$

$$u_{ik} + 1 \leq u_{jk} + M(1 - x_{ijk}), \forall (i,j) \in A\colon i \neq 0, j \neq 0, \forall k \in K \tag{7}$$

$$d_i x_{ijk} \leq u_{ik} \leq Q_k, \forall (i,j) \in A\colon i \neq 0, j \neq 0, \forall k \in K \tag{8}$$

$$x_{ijk}, z_{ik} \in \{0,1\},\ \forall (i,j) \in A, \forall i \in V \setminus \{0\}, \forall k \in K \tag{9}$$

In the formulation above, at first the objective function is represented. It aims to minimize the sum of the costs of the routes traveled by each vehicle, corresponding to its type. After that, all the essential constraints are listed, such that, (1) ensures flow conservation, while (2) makes sure that all tours begin and end at the depot, ensuring that every vehicle makes at most 1 tour. Constraints (3) and (4) guarantee that all customer demand is satisfied without violating vehicle capacity. Constraint (5) and (6) force the binary variables to be 1 when there is any amount delivered to a certain customer, therefore activating the correspondi ng arcs in x_{ijk}, then, (7) and (8) are slightly reformulated versions of the Miler-Tucker-Zemlin constraints, which assign a value u to each customer that changes after each delivery and ensures that the vehicles don't drive in circles and while also avoiding subtours. Finally, (9) states the binary nature of variable x_{ijk} and z_{ik}.

2 Description of the instances

In order for an instance to have at least one feasible solution, it needs to include the following input data:

1) Number of customers
2) Demand from customers
3) Maximum capacity for each type of vehicle
4) Supply for each type of vehicle
5) Cost matrix based on distance between nodes and cost factor for each type of vehicle

To obtain feasible instances, 3 different sets with varying scales and degrees of complexity were extracted and adapted from the available literature, comprising in total 15 different instances to solve. The first of them is a set of 5 small-scale instances CHU-01,..05, obtained from Chu [3], then 5 medium-scale ones, CE-01,...05 from Christofides & Eilon [2]. Finally, 5 large-scale instances G-01,...05 from Golden et al [7]. These isntances can be seen in Table 1.
Since the instances were originally intended for heterogeneous fleets no adaptation was needed, however, to accommodate for the fact that for this problem the small-size vehicle can only carry half of the maximum capacity Q_k of the large-size vehicle, the demand of customers had to be adapted to be able to be satisfied with the new capacities so that total demand $\sum d_i$ would not exceed $\sum S_t Q_k$.

Instance	Customers	Demand	Capacity		Supply	
			LargeVehicle	SmallVehicle	LargeVehicle	SmallVehicle
CHU-01	5	7:10	30	15	1	1
CHU-02	10	7:10	70	35	1	1
CHU-03	15	7:10	80	40	1	2
CHU-04	20	7:10	70	35	2	2
CHU-05	25	7:10	75	37.5	2	3
CE-01	40	12:15	400	200	1	1
CE-02	50	12:15	170	85	3	3
CE-03	60	12:15	140	70	5	3
CE-04	75	12:15	210	105	3	5
CE-05	100	12:15	250	125	5	3
G-01	120	5:10	800	400	1	1
G-02	140	5:10	560	280	2	1
G-03	160	5:10	650	325	2	2
G-04	180	5:10	600	300	3	1
G-05	200	5:10	585	293	3	2

Table 1: Characteristics of Instances to solve

2.1 Randomly generating location of customers

With regards to the distances and cost matrix, a similar approach of that implemented in Bolduc et al. [4] was used, using the city of Pisa as reference, a radius of 4 km has been drawn around the city, encompassing an area of around 100 km^2 (Figure 1). This area is intended to serve as a delivery area where the depot is located at the center and the location of each n customer is to be randomly distributed within the area.

Figure 1: Delivery area

For the random location of customers, distances were generated using the Python function random.randint() from the NumPy library. The same function was used to determine the €/km cost factor, between the range of 0.5 to 1 to get the total cost of traveling from point i to j and j to i. An example of this process applied to instance CHU-01 can be seen in figure 2 and table 2, where fig. 2 is a representation of the area in fig. 1, showing some of the possible pairs of distances between nodes and demonstrating their asymmetry, owing to real-case scenarios such as one way streets or highways which can take long deviations to reach a certain customer, e.g. $d_{02} = 4.52$ km while $d_{20} = 4.03$ km. Table 2 shows every distance in km between possible pair of nodes.

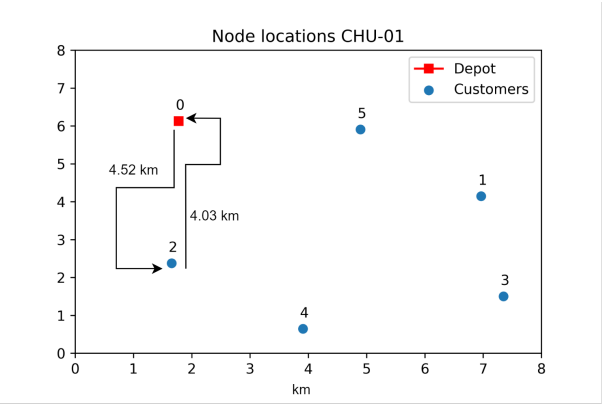

Figure 2: Delivery area for CHU-01

	0	1	2	3	4	5
0	0	6.17	4.52	7.76	6.18	3.31
1	5.63	0	6.34	3.11	4.81	3.60
2	4.03	6.01	0	6.06	3.47	5.37
3	7.84	2.94	6.05	0	3.80	5.37
4	6.02	4.81	3.80	4.51	0	5.54
5	3.15	2.92	5.49	5.82	5.37	0

Table 2: d_{ij} for CHU-01

Now having the distances between nodes, the cost matrix can be easily calculated using the previously obtained random cost factor, in the case of CHU-01 being equal to 0.8, c_{ij}^t for t1 and t2 can be seen in table 3 and 4 respectively.

	0	1	2	3	4	5
0	0	5.55	4.07	6.98	5.56	2.98
1	5.07	0	5.70	2.80	4.33	3.24
2	3.63	5.41	0	5.45	3.12	4.84
3	7.06	2.64	5.44	0	3.42	4.83
4	5.42	4.33	3.42	4.05	0	4.99
5	2.83	2.63	4.94	5.24	4.84	0

Table 3: c_{ij}^1 for CHU-01

	0	1	2	3	4	5
0	0	4.16	3.05	5.24	4.17	2.24
1	3.80	0	4.28	2.10	3.24	2.43
2	2.72	4.06	0	4.09	2.34	3.62
3	5.29	1.98	4.08	0	2.56	3.62
4	4.07	3.25	2.57	3.04	0	3.74
5	2.13	1.97	3.71	3.93	3.63	0

Table 4: c_{ij}^2 for CHU-01

3 Initial implementation

In order to have a first implementation, the use of the Gurobi solver with its Python Interface has been preferred due to its efficient and comprehensible environment that offers state-of-the-art algorithms for solving MILP models. For this scope, a .ipynb notebook was created importing the Gurobi Solver with version 9.5.2 and build v9.5.2rc0, using an academic licence.
For this initial implementation, only the basic model was built with its objective function and constraints, without making use of advanced techniques or changing the default parameters, this in order to better appreciate the improvements implemented in further sections. Initially, an optimization run was computed up until finding the first feasible optimization. To see how much it will differ to the solutions found after.

As we can see in figure 3 and 4, the 1st and optimal solution present several differences, starting from a very different objective to a route that is clearly better as seen in the plot of the optimal solution. Another thing to notice is how, at least in the first instances, the optimal solutions are much more understandable to the human eye in comparison to the first solutions, which almost seem to make no sense. This can be seen in figure 5 and 6.

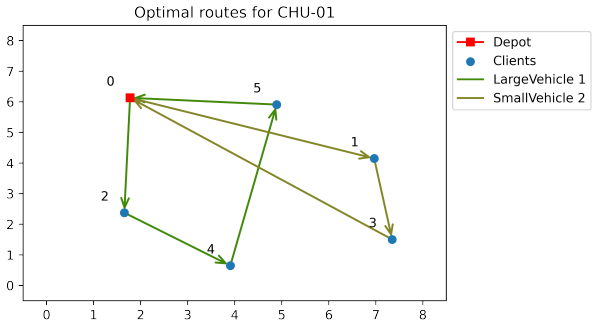

Figure 3: Optimal routes for CHU-01

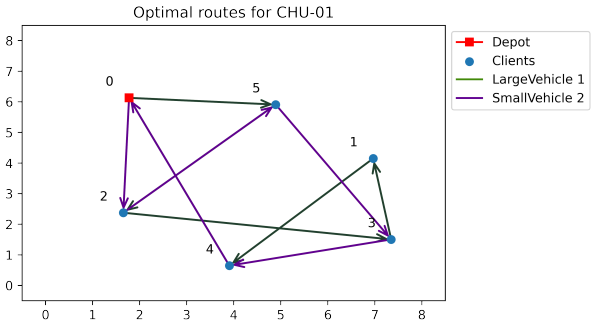

Figure 4: 1st solution found for CHU-01

Objective: 22.56 €
k1: $0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 0$
k2: $0 \rightarrow 1 \rightarrow 3 \rightarrow 0$

Objective: 39.5 €
k1: $0 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 0$
k2: $0 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 0$

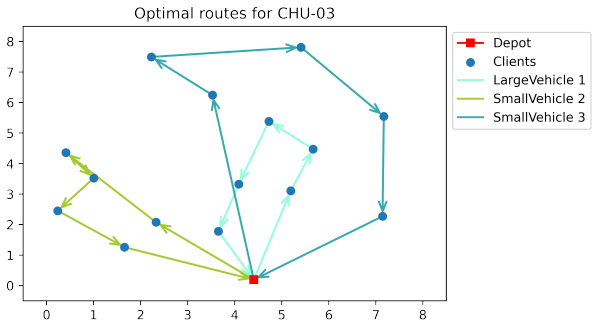

Figure 5: Optimal routes for CHU-03

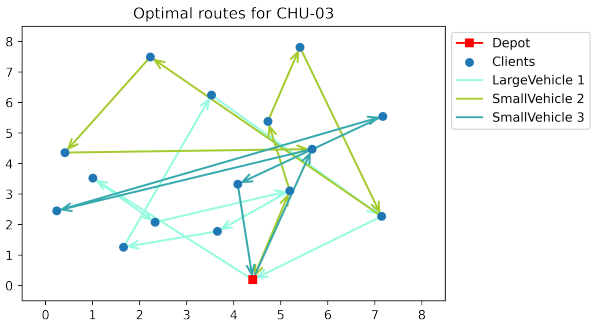

Figure 6: 1st solution found for CHU-03

Objective: 33.24 €
k1: $0 \rightarrow 12 \rightarrow 1 \rightarrow 15 \rightarrow 3 \rightarrow 11 \rightarrow 0$
k2: $0 \rightarrow 2 \rightarrow 8 \rightarrow 6 \rightarrow 10 \rightarrow 7 \rightarrow 0$
k3: $0 \rightarrow 9 \rightarrow 13 \rightarrow 14 \rightarrow 5 \rightarrow 4 \rightarrow 0$

Objective: 64.91 €
k1: $0 \rightarrow 6 \rightarrow 2 \rightarrow 12 \rightarrow 11 \rightarrow 7 \rightarrow 9 \rightarrow 4 \rightarrow 0$
k2: $0 \rightarrow 12 \rightarrow 15 \rightarrow 14 \rightarrow 4 \rightarrow 13 \rightarrow 8 \rightarrow 0$
k3: $0 \rightarrow 1 \rightarrow 10 \rightarrow 5 \rightarrow 3 \rightarrow 0$

Then as we move up in complexity, starting with the set of instances CE, optimality is no longer reachable in a reasonable time, nonetheless, there is still a clear structure to be seen in the assignment of routes as seen in figure 7. However, when solving for instances G, where the amount of customers is greater than 100, this structure is lost, which can be seen in figure 8.

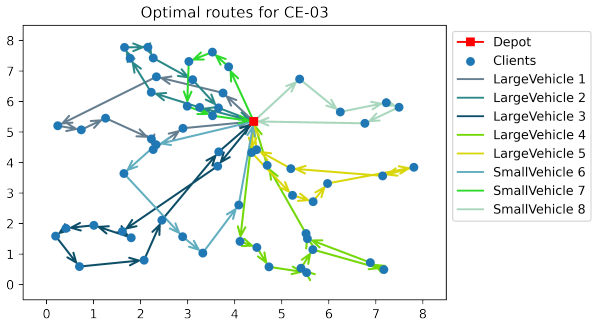

Figure 7: Best tours for CE-03

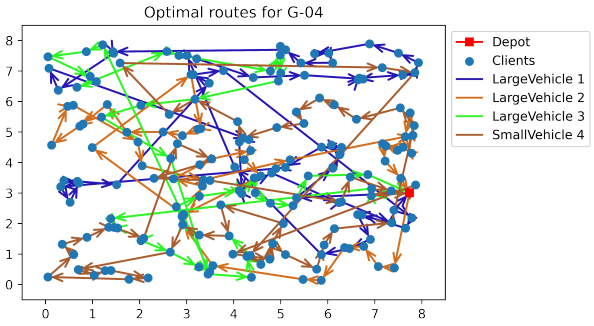

Figure 8: Best tours for G-04

4 First computational experiences

After building the MILP models for each of the instances of table 1, they were solved using a portable computer running Windows 10 home 64bit, 8GB of RAM and processor intel i5-10300H 2.50GHz using 4 cores and 8 logical processors. It is worth noting that different time limits were imposed for each set of instances, being 1,200, 3,600 and 7,200 seconds respectively.

One of the most significant sources for computational results can be obtained by the log file provided by the solver, which produces a detailed description of the optimization process, it is divided into three sections, the first of it being "Presolve", where a presolving algorithm is applied in order to try to simplify the model by removing rows and columns and then passing this new model to the simplex solver. [Basically, what is done before the branching process is started.] An interesting thing to note here is that this section removed mostly rows and not columns for the most part, which is due to the fact that our model contains more constraints than variables, ending up in models with roughly double the rows, for example in CHU-05, the model starts with 6545 rows and 3625 columns and after finishing the presolve section, it passes 3540 and 3625 respectively to the simplex solver.

The process is then followed by "Progress", which tracks the activities of the branch-and-cut search, it begins with the root relaxation, providing a bound on the best possible objective. Then it displays the progress of the search in a quantitative way, showing the number of nodes that have been explored in the search tree and those unexplored. It also shows the objective of the current node being explored and its depth in the tree.

The log also provides the objective bounds, containing the primal objective bound, i.e. the incumbent objective, which acts as an upper bound for minimization problems such as ours, this is the objective of the best known feasible solution. [It is done during the "Presolve" phase, when first heuristic solution is set as upper bound (first incumbent), so that our solver knows that it can take at most that value in further iterations.] Then we have the dual objective bound, i.e. the lower bound, giving a bound on the best possible objective. [Considering that we are dealing with minimization problem, this initial lower bound i.e. best bound is provided at the first stage of LP relaxation.]

Additionally, the log shows the runtime elapsed since the solver began. Then it ends at the "summary" section. which shows the total runtime, amount of nodes explored, simplex iterations, solution count and its objectives as well the best objective bounds.

The key features obtained from the previous log can be seen in table 5, where we report the following elements: optimality (tolerance 1.00e-04), runtime of both the 1st and best solution

found and upper and lower bounds for both solutions with the gap between these bounds. In addition to the table, figure 9 and 10 show the graphical difference in runtime and objective for both solutions.

[The process from the first to the best solution is the following. Considering that we are dealing with minimization problem the primal CPLEX provides us the value of some feasible solution that is above the optimal value. Step-by-step it generates a little closer value to the optimal, so that the solver constructs the sequence of decreasing values of the objective until it gets to it. But we also need to have some value there below the optimal one to know how close approximation is the current primal solution. The dual is that lower bound, that allow us to evaluate how close our feasible solution is. The duality helps us to immediately understand when we have reached optimality, simply when the primal has the same value as dual then we know it is optimal. In the log file we can keep track of upper and lower bound values, as well as the gap between them at each iteration(?). We can say that the high value of gap is mostly due to the upper bound which tends to be much higher than the possible optimal solution in most instances(? my opinion, not sure).]

As can be seen from the table, only 4 instances attained optimality within the time limits, 3 from set CHU and 1 from set CE, this shows the high amount of complexity that the different number of vehicles add to the model, taking for example CE-01 which was able to reach optimality even with 40 customers but only 1 vehicle of each type, whereas CHU-04 with only 20 customers was not able to do it due to having 2 vehicles of each type. Due to this, a second run of optimization was performed in order to achieve near-optimality, but this time with the addition of the parameter MIPGap, which sets a minimum threshold for the gap. Together with the time limits, this sets a limit of either time or gap to terminate the optimization.
This sudden and exponential increase in complexity can also be seen in figures 9 and 10, where starting from CHU-04, the runtimes and objective values begin to go up exponentially.

Instance	Optimality	Runtime		1st ObjBounds			Best ObjBounds		
		1st	Best	Upper	Lower	Gap	Upper	Lower	Gap
CHU-01	Yes	0.0129	0.0409	39.50	0.00	100%	22.55	22.55	0%
CHU-02	Yes	0.0150	0.3723	57.84	0.00	100%	29.20	29.20	0%
CHU-03	Yes	0.0698	8.2341	64.91	27.25	58%	33.24	33.24	0%
CHU-04	No	0.0459	680.08	124.8	0.00	100%	35.86	30.48	14.9%
CHU-05	No	0.0879	1093.29	187.7	0.00	100%	49.56	42.97	13.3%
CE-01	Yes	0.7049	1748.14	109.2	39.40	63.9%	42.86	42.86	0%
CE-02	No	0.3971	2736.2	289.1	0.00	100%	73.41	51.80	29.4%
CE-03	No	1.4572	3183.5	367.4	57.01	84.4%	75.80	63.02	16.8%
CE-04	No	1.2776	3442.8	470.8	0.00	100%	106.75	66.49	37.7%
CE-05	No	10.339	3564.3	551.7	70.11	87.2%	112.67	75.67	32.8%
G-01	No	23.357	6627.7	469.1	69.49	85.1%	93.07	72.46	22.1%
G-02	No	143.83	6517.9	261.5	83.29	68.5%	111.58	88.51	20.6%
G-03	No	1234.26	5224.1	660.2	89.04	86.5%	135.09	90.40	33.1%
G-04	No	2782.97	7157.2	1027.5	100.32	89.9%	201.22	100.4	48.1%
G-05	No	4954.18	7200.4	864.9	100.90	87.3%	471.77	100.99	76.7 %

Table 5: First computational results

As seen in figure 9, the runtime needed to get the first feasible solution is quite small and almost instantaneuous for most instances apart from G-03 on, where it dramatically increases up to 5000 seconds, the best solutions (within the time limit) on the other hand, follow a less uniform path but overall adhere to the same trend, with the main difference being that at the

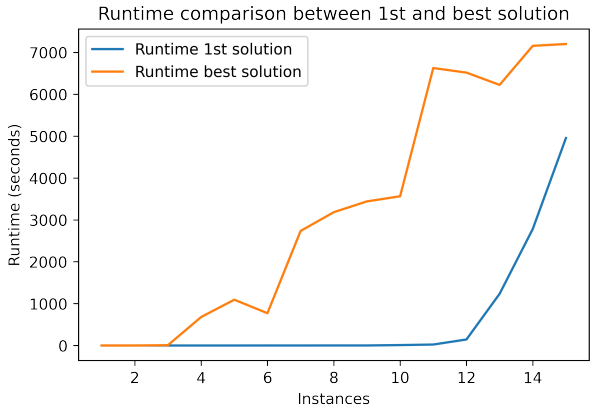

Figure 9: Runtime comparison between 1st and best solution

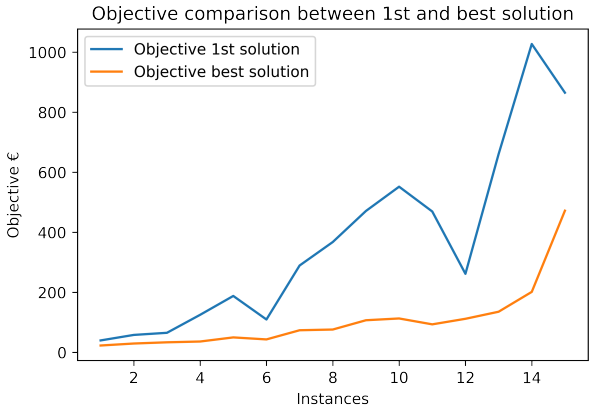

Figure 10: Objective comparison between 1st and best solution

later stages of the optimization process, many instances progress actually stalled, providing no new bounds, so the results remained unchanged for prolonged periods of time, in figure 9 the runtime reported is the one seen at the first appearance of the best solution. Figure 10 follows the same trends, except for instance G-02, where the first solution managed to find a very low objective value compared to remaining instances in the G set.

5 Proposed improvements

5.1 Warm-Starting

The first approach proposed to improve performance is warm-starting, also known as advanced starting, which considerably reduces computational time by providing the solver with an already known feasible solution obtained from a previous optimization run, in our case, we used the first available solution obtained in the previous sections and retrieved its .sol file to feed into the next iteration of the solver. With this approach we can skip the time spent looking for a feasible solution. By applying this approach, we can see not only a reduction in computation time but also better feasible solutions and smaller gaps due to the increased amount of time available for the solving process.

5.2 Parameter Tweaking

Presolve phase
Within the presolve phase of the solution we have a set of parameters that directly influence the presolving and can lead to a smaller and tighter formulation and therefore leading to a more efficient subsequent branch-and-bound. **PrePasses** provides finer-grain control by limiting the number of passes presolve performs, which can reduce its runtime if it becomes an issue. The **AggFill** parameter controls aggregation at a finer grain. Aggregation typically leads to a smaller formulation, although in rare cases it can introduce numerical issues. The **PreSparsify** parameter enables an algorithm that can sometimes significantly reduce the number of non-zero values in the constraint matrix.

MIPFocus
Allows the user to adjust the solving strategy for MIP models depending on the goals. By default, Gurobi strikes a balance between finding new feasible solutions and proving that the current solution is optimal, but this focus can also be put on finding good quality feasible solutions, on proving optimality or on the best objective bound, depending also on how the solver

is behaving for a particular model.

Heuristics
The **Heuristics** parameter controls the fraction of runtime spent applying feasibility heuristics. Higher values can lead to more and better feasible solutions, but at the cost of rate of progress in the best bound. The **SubMIPNodes** parameter controls the number of nodes explored in the local search heuristics, a higher value gives better solutions but at computational cost. There are also heuristics like NoRel heuristic (controlled by the **NoRelHeurTime**) that attempt to find high-quality solutions before solving the MIP relaxation. This can be quite effective to reduce the initial time spent on the root node of the tree and provide better feasible solutions early on, improving computational time.

Cutting planes
With the **Cuts** parameter, the aggressiveness of the cutting strategy can be controlled on a general scale, affecting every type of cut, and at a finer grain one can specify the aggressiveness of a particular type of cut, e.g. **FlowCoverCuts** or **MIRCuts**. Larger models such as the ones in our G instances benefit from a more aggressive cutting strategy to reduce their complexity.

In order to find the parameters that best suit a model, two approaches were used, first, in an intuitive way, suitable parameters were chosen for each particular case, for example following the case of CHU-04, after looking at the log of the base model solution, we can see how most of the convergence between upper and lower bounds is happening within the first 100 seconds, after this, no new incumbent is found and the lower bound increases ever so slightly, arriving at a point where it barely moves anymore, reaching the time limit. A first attempt to fix this is to focus on the upper bound, trying to put more attention on the branching part and finding more feasible solutions, through the use of MIPFocus, cuts and heuristics, we get very little to no changes in the incumbent solution, hinting that the problem instead relies on the lower bound, so we find the parameters more suited to tackle that issue.
The second approach is provided by gurobi in the form of a tuning method, which automatically analyzes the performance of different combination of parameters on small portions of the solving process and returns those with the best results. This however is extremely costly and didn't give back better results as compared to picking the parameters by hand.

5.3 User Cuts - Lazy Constraints

Although slightly similar in purpose, user cuts and lazy constraints are a pair of distinct set of tools that can potentially improve the performance of a MIP model. Cuts are constraints that can change the feasible space of the continuous relaxation but do not rule out any other feasible integer solution allowed in the model. Essentially, the formulation remains the same whether the cuts are added or not. Lazy constraints on the other hand are simply constraints that are a needed part of the formulation but have adopted a "lazy" approach, that is, being checked only after identifying a particular integer solution candidate that violates this constraint, therefore saving in computational time when they are not needed.

In figure 11 we can see the actual difference these approaches produce on the outcome of the optimization. Whereas the first solution used all vehicles due to constraint (2), by using user cuts, we can relax this constraint and allow the solution to use fewer vehicles than needed as long as it doesn't violate any other constraint, as a result, we can actually obtain an optimal solution with objective 35.14.

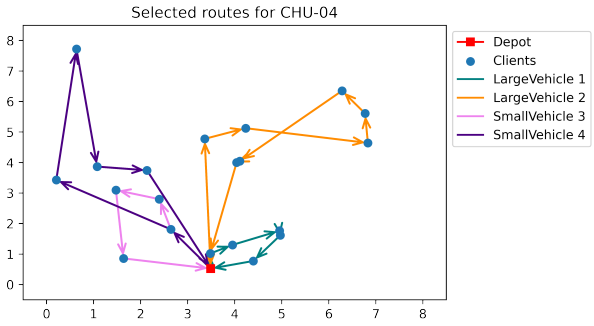

Figure 11: CHU-04 before user cuts

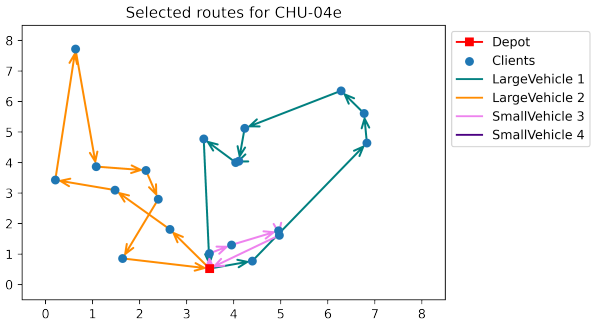

Figure 12: CHU-04 after user cuts

5.4 FeasRelaxation - Artificial variables

By adding artificial variables to each constraint and then minimizing their sum we can obtain a solution with objective zero, which corresponds to a feasible solution to the input model. Then using FeasRelax we optimize the returned model, which minimizes the original objective, but only from among those solutions that minimize the sum of the artificial variables. FeasRelax basically creates a feasibility relaxation, that when solved, minimizes the amount by which the solution violates the bounds and linear constraints of the original model. This technique proved to speed up computation times, for example in CE-01, where it previously needed 1748 seconds to reach optimality, now with the feasibility relaxation, it only needs 819.

5.5 Lagrangian Relaxation

Lagrangian Relaxation (LR) has also proved to be a powerful technique to improve the performance of our solving process, especially on the latter part of the G instances, where it was able to reduce the gap by more than double in several occasions. The fundamental purpose of LR is to try to include "hard" constraints in the objective function, multiplied by the Lagrangian multiplier λ, and as a result reduce the number of "hard" constraints in the problem formulation that makes it much easier to solve. This technique should provide better lower bounds (for minimization) and quicker feasible solutions. In our case, after some empirical experimentation, we found constraint (2) to be the most beneficial to use as the separated constraint to add to the objective function together with λ. Although highly effective for some instances, it added little to no benefit for other instances, such as those of the CE group for example.

5.5.1 Proposed Reformulation of a MILP Model

Minimize

$$\sum_{(i,j)\in A}\sum_{k\in K1} c^1_{ij}x_{ijk} + \sum_{(i,j)\in A}\sum_{k\in K2} c^2_{ij}x_{ijk} + \sum_{j\in V\setminus\{0\}}\sum_{k\in K} \lambda_{jk}(1 - x_{0jk})$$

The above is the LR reformulation of our model's objective function. We only represent the objective function which is changed in terms that we have the dual of constraint (2) added to it, multiplied by λ.

6 Evaluating the effect of the proposed improvements

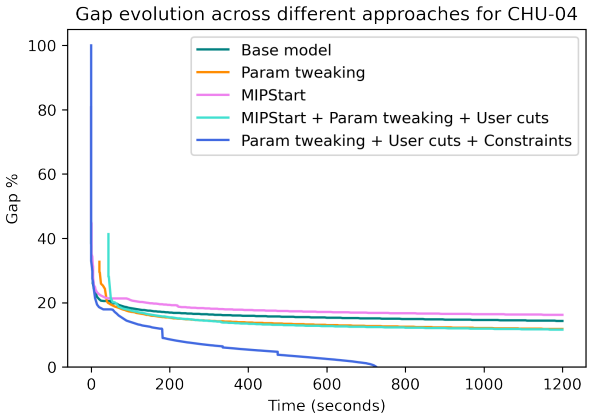

Figure 13: Gap evolution across different approaches to solve CHU-04

As shown by figure 13, by seeing the graphical comparison of the evoluton of gap among the different approaches tried to improve for example CHU-04, we can see a clear trend of most of the solving happening during the first stages of the process, then stabilizing and hardly changing for the remainder of the computing time. Knowing this, we can infer that it is highly important to implement improvements that can refine the search region, therefore providing better and quicker feasible solutions while also arriving at acceptable lower bounds. Following the example of CHU-04, this is done by adding a set of extra inequalities, which essentially act as a cutting plane, tightening the formulation and removing undesirable fractional solutions. With this, we can appreciate a rapid solution and declining of the gap in figure 13 for the approach with user cuts and extra constraints.

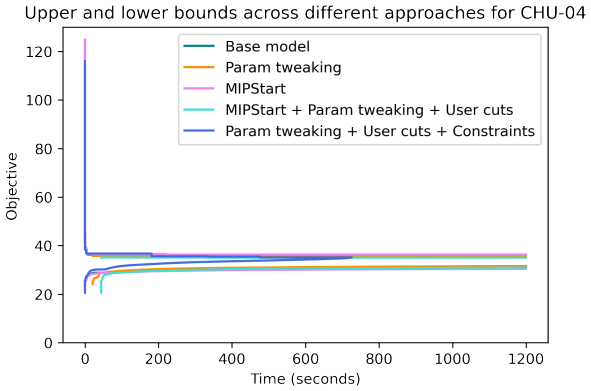

Figure 14: Upper and lower bounds across different approaches to solve CHU-04

Another interesting thing to gather from the behavior of the upper and lower bounds depending on the improvement technique used is seen in figure 14, where we can appreciate a constant trend of both bounds after the first minutes of computation, stabilizing and not converging, ending up with a considerable gap, however, after seeing the line corresponding to the optimal solution, we can see that it was in fact the lower bound to be blamed for the gap, as the optimal gap was mostly reached by most solving methods, confirming the hypothesis presented in section 5.2 about the need for tightening techniques such as the new cutting planes, or a more aggressive presolving which could have also worked.

Table 6 summarizes the results for every instance and will be visually complemented with figures 15 to 18 to piece together the whole picture and get an understanding of the behaviors of each instance and the optimization techniques here presented.

Instance	Opt	Runtime		Previous ObjBounds			Improved ObjBounds		
		Prev	Impr	Upper	Lower	Gap	Upper	Lower	Gap
CHU-01	Yes	0.0409	0.0319	22.55	22.55	0%	22.55	22.55	0%
CHU-02	Yes	0.3723	0.2253	29.20	29.20	0%	29.20	29.20	0%
CHU-03	Yes	8.2341	7.1089	33.24	33.24	0%	33.24	33.24	0%
CHU-04	Yes	1170	725.95	35.93	30.79	14.3%	35.14	35.14	0%
CHU-05	No	1127.2	1196.1	48.97	43.51	11.2%	46.43	45.07	3.06%
CE-01	Yes	1748.14	819.35	42.86	42.86	0%	42.86	42.86	0%
CE-02	No	2736.2	3198.8	73.41	51.80	29.4%	67.52	52.43	22.27%
CE-03	No	3183.5	3374.3	75.80	63.02	16.8%	75.28	63.37	15.81%
CE-04	No	3442.8	3442.8	106.7	66.49	37.7%	106.7	66.86	37.4%
CE-05	No	3564.3	3562.3	112.7	75.67	32.8%	109.8	75.18	31.56%
G-01	No	6627.7	6199.3	93.07	72.46	22.1%	84.91	72.37	14.76%
G-02	No	6517.9	4765.8	111.6	88.51	20.6%	102.6	88.63	13.63%
G-03	No	5224.1	6183.5	135.1	90.40	33.1%	118.8	90.09	24.18%
G-04	No	7157.2	7189.1	201.2	100.4	48.1%	131.9	103.9	21.21%
G-05	No	7200.4	6888.9	471.8	100.9	76.7 %	155.4	108.7	30.0%

Table 6: Computational results after applying proposed improvements

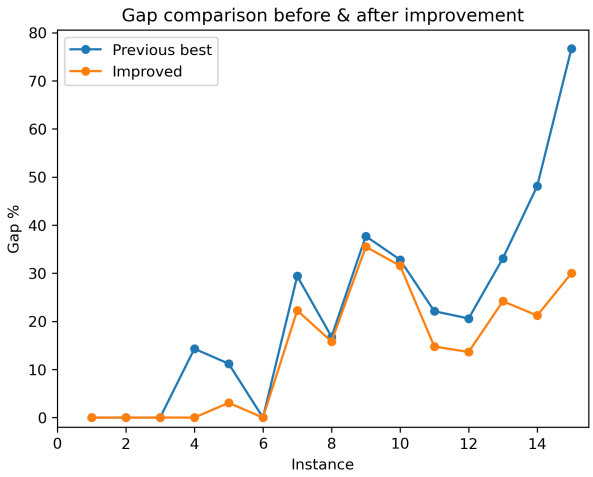

Figure 15: Gap across instances

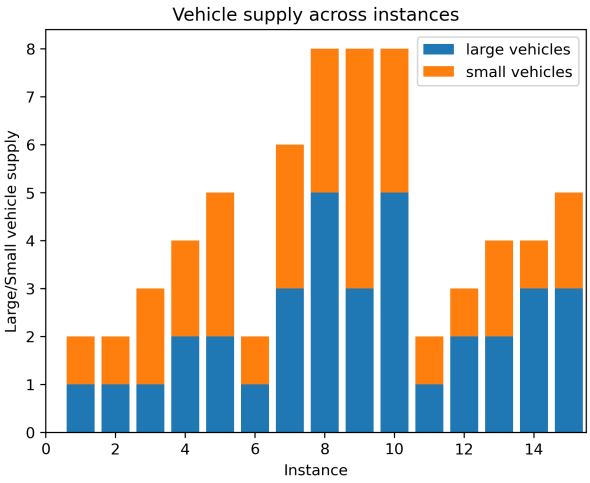

Figure 16: Vehicle supply across instances

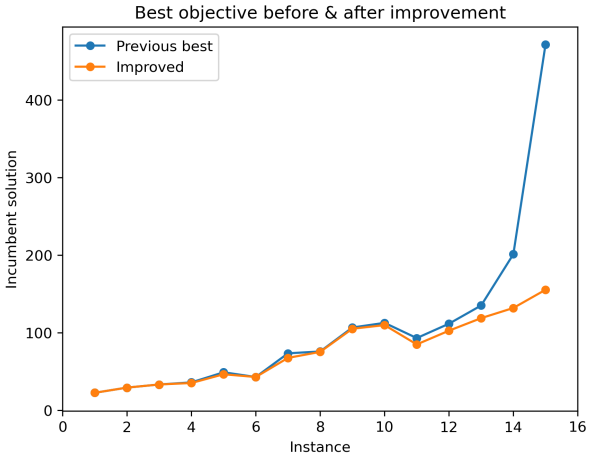

Figure 17: Objective comparison across instances

Ultimately, having obtained the best possible solutions, most of the instances from the first set were able to be solved to optimality or near, greatly improving in contrast to the solving without improvement techniques. Moving on to the CE set, where the complexity increases both in number of clients to be served as well as the available number of vehicles to deliver, the improvements were less noticeable, and as seen in figure 15, the gap hardly moved, even when using powerful techniques such as LR. Finally with set G, we can observe the biggest improvements, thanks mostly to LR. Interestingly, even though the G set has a lot more clients to be served, the gaps and best objectives were consistently and counter-intuitively lower than those of CE, showing that it is actually the number of vehicles the determining factor to ease of computation for this problem. Shown in figure 16, we can see how the number of vehicles corresponds to the results previously seen. We can then see the same behavior in figure 17 with the best objectives found, where the follow a mostly linear trend corresponding to the size of the instance, except for the CE set, where it is probable that the feasible solutions found were not very close to optimal.

7 References

(1) Bianchessi, Nicola & Drexl, Michael & Irnich, Stefan. (2017). The Split Delivery Vehicle Routing Problem with Time Windows and Customer Inconvenience Constraints.

(2) Bigi, G., A. Frangioni, G. Gallo, and M.G. Scutellà (2014). Appunti di Ricerca Operativa.

(3) Bolduc, Marie-Claude & Renaud, Jacques & Montreuil, Benoit. (2006). Synchronized routing of seasonal products through a production/distribution network. Central European Journal of Operations Research. 14. 209-228. 10.1007/s10100-006-0169-2.

(4) Christofides, N. & Eilon, S. (1969). An algorithm for the vehicle-dispatching problem, Oper. Res. Q. 20 309–318.

(5) Chu, C.W. (2005). A heuristic algorithm for the truckload and less-than-truckload problem, Eur. J. Oper. Res. 165 657–667.

(6) Euchi, Jalel. (2017). The vehicle routing problem with private fleet and multiple common carriers: Solution with hybrid metaheuristic algorithm. Vehicular Communications. 9. 97-108. 10.1016/j.vehcom.2017.04.005.

(7) Euchi, Jalel & Chabchoub, Habib. (2022). Heuristic Search Techniques to Solve the Vehicle Routing with Private Fleet and Common Carrier.

(8) Golden B.L., Wasil E.A., Kelly J.P., Chao I.M. (1998). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem tests and computational results, in: T.G. Crainic, G. Laporte (Eds.), Fleet Management and Logistics, Kluwer, Boston, pp. 33–56

(9) Hasibuan, H & Suwilo, S & Sitompul, Opim. (2019). Fleet Size from Split Delivery in Vehicle Routing Problems. Journal of Physics: Conference Series. 1255. 012039. 10.1088/1742-6596/1255/1/012039.

(10) Herrero, Rosa & Villalobos, Alejandro & Caceres Cruz, Jose & Juan, Angel. (2014). Solving Vehicle Routing Problems with Asymmetric Costs and Heterogeneous Fleets. International Journal of Advanced Operations Management. 6. 58-80. 10.1504/IJAOM.2014.059620.

(11) Herrero, R. (2015). Hybrid Methodologies for Symmetric and Asymmetric Vehicle Routing Problems. Doctoral Thesis. Universitat Autònoma de Barcelona

(12) Li, Yinglei, "Robust vehicle routing in disaster relief and ride-sharing: models and algorithms" (2017). Graduate Dissertations and Theses. 49.

(13) Maheo, Arthur & Urli, Tommaso & Kilby, Philip. (2016). Fleet Size and Mix Split-Delivery Vehicle Routing.

(14) Moshe Dror, Gilbert Laporte, Pierre Trudeau, Vehicle routing with split deliveries, Discrete Applied Mathematics, Volume 50, Issue 3, 1994, Pages 239-254, ISSN 0166-218X, https://doi.org/10.1016/0166-218X(92)00172-I.

(15) Ragsdale, C.T. (2004). Spreadsheet Modeling and Decision Analysis: A Practical Introduction to Management Science. Cengage South-Western.

(16) Wilck, Joseph & Cavalier, Tom. (2014). A Column Generation Procedure for the Split Delivery Vehicle Routing Problem Using a Route-Based Formulation. International Journal of Operations Research and Information Systems. 5. 44-63. 10.4018/ijoris.2014100103.

(17) Wilck, Joseph & Cavalier, Tom. (2012). A Construction Heuristic for the Split Delivery Vehicle Routing Problem. American Journal of Operations Research. 02. 153-162. 10.4236/ajor.2012.22018.

(18) Scutellà, M.A. (2015). Logistics, lecture Notes.