Solving the Vehicle Fleet Mix Problem (VFMP) with Route-First Cluster-Second Heuristics

Tomas Miskov

2022-10-26

Introduction

Given a fleet of company vehicles and a set of customers, what is the least cost intensive routing of the vehicles that serves all the customers? This problem is the most elementary version of the vehicle routing problem (VRP). Given a set of vehicles, a central location (depot), a road network, and a set of customers, the VRP solution provides a set of routes that minimizes the global traveling costs. Beyond its basic formulation, many variants have been proposed and subsequently solved using a variety of methods in the literature. In this report, we focus on the Vehicle Fleet Mix Problem (VFMP) as first described in Golden et al. (1984).

VFMP is an extension on the capacitated VRP (CVRP) where every customer has a demand quantity and vehicles have a maximum load they can carry. In CVRP, all vehicles are the same, thus the routes are only constrained by a single maximum capacity constraint. The VFMP extends this problem to a heterogeneous fleet of vehicles with different capacities and costs (both fixed and/or variable) for every vehicle. The number of vehicles per vehicle type is not restricted, thus in real-world setting, the VFMP solves the problem of determining the optimal fleet of vehicles to be purchased for operation. As described in Prins (2009), the historical record of VFMP problems looks into three sub-variants, namely the VFMP-F, VFMP-V, and VFMP-FV, where F stands for fixed costs, V for variable costs, and FV for fixed and variable costs. In this report we focus on solving the VFMP-F, since the test instances considered only include the fixed costs and the maximum capacity constraint for every vehicle type. The instances are extensions on the synthetic data generated by Christofides and Eilon (1969). Every instance includes the following information:

- set of customers with their locations as a tuple of (x, y) coordinates in the 2D plane
- set of customer demands as an integer value specifying the demanded quantity
- depot and its location
- set of vehicle types with corresponding capacities and fixed costs, both integer valued

Given such an instance, the solution consists of determining the number of vehicles of each vehicle type and their corresponding routes over a subset of customers. For every solution, we can compute the corresponding cost, which is just the summation of the fixed costs over all the purchased vehicles, and the summed trip mileage over all the proposed routes. Comparing this cost to the best known solution obtained in the literature, we get an optimality gap, that determines the strength of our proposed algorithm. The following section describes the implemented algorithms and the details of the implementation. Afterwards, the experiments are described, and their results presented. Finally, a short conclusion on the observed results is offered, and potential improvements to the algorithms are discussed.

Methods

In general, the VRP is an NP-hard problem, which in laymen terms just means *very hard*. A more precise definition is that not all instances of VRP can be solved in polynomial time. Hence, exact combinatorial

optimization methods such as constrained linear programming, can only solve instances of a limited size to optimality. For this reason, to solve the VFMP-F problem, we will use a number of heuristic methods that aim to provide a good solution, but cannot prove optimality. A general approach applied in this project is the **route-first cluster-second** methodology introduced by Beasley (1983). Under this methodology we start by constructing a a so-called *giant tour* over all the customers and the depot. Given a giant tour, we proceed to split it, creating individual trips for our fleet of vehicles. This can be followed by a number of improvement methods commonly dubbed as local search, or the whole algorithm can be parametrized and run in an iterative and multi-start fashion, generating the least costly set of routes and vehicles found over all the obtained feasible solutions. The remainder of this section thus first discusses the giant tour construction followed by the splitting algorithm, the local search methods, and the extension to the iterative and multi-start approaches.

Giant Tour Construction

Constructing a single route over all the customer nodes and the depot reduces the initial VRP problem to a Traveling Salesman Problem (TSP). Alongside a number of exact methods that can solve very large TSP instances, up to thousands of nodes, to optimality, many heuristic algorithms for generating good TSP solutions exist. Since an optimal giant tour does NOT necessarily lead to an optimal VRP (or VFMP) solution, it is sufficient and even beneficial to construct just *good* TSP solutions. The benefit lies in the fact that many good tours can be constructed, yielding in turn (hopefully) many good solutions to the VFMP problem. The term *good* refers to close to optimal solutions.

In this paper we implement the k-nearest neighbor (knn) algorithm for constructing the giant tour. Starting at some node i, the knn algorithm constructs a giant tour by iterativelly considering k nearest neighbors of the last visited node, picking one uniformly at random as the next node to be visited. If $k = 1$ and the starting node is fixed, the algorithm is deterministic. Otherwise there is $k^{(n+1)}$ possible routes for every starting node, where n is the total number of customers and $+1$ is the depot . Thus, the two parameters of the algorithm are k, the number of neighbors to consider at each iteration, and i the initial starting node. Often the depot is chosen as the starting node, however, it is not necessary to start with the depot, and as shown in the results, initializing the tour from different nodes often provides improving solutions. The order of running time of the knn giant tour algorithm is $\mathcal{O}(n+1)$, as the tour is constructed in a single loop, over all the nodes.

Optimal Splitting

To obtain a solution to the VRP/VFMP from an initial giant tour, we need an algorithm that efficiently partitions the giant tour into individual trips and assigns vehicles to them. To do this we implement an extension of the well-known *split* algorithm proposed by C. Prins (2009). Compared with the CVRP with a homogeneous fleet of vehicles, *split* for VFMP-F relies on two notable extensions. First, the maximum capacity of each trip is only constrained by the capacity of the largest vehicle in our fleet. And second, an additional data structure is created to link every possible trip load to a vehicle type. Thus, given a set of vehicle types T and a set of their corresponding capacities T_c, the maximum capacity Q equals to max (T_c). Then, a dictionary (*Load2vehicle* in Algorithm 1) is created with keys $\{0, 1, 2, ..., Q\}$ and values corresponding to the fixed costs of the smallest vehicle type $t \in T$ that can still carry a load defined by its key.

As in the original CVRP *split* algorithm we construct an implicit auxiliary graph with $n+1$ nodes and an arc for every feasible trip. Since such graph is acyclic, a shortest path from node 0 to node n can be found. The arcs of this shortest path then define the trips that should be made since every arc corresponds to a feasible route in the original VFMP-F. Algorithm 1 provides a pseudo-code that closely resembles our implementation in Python. Because the auxiliary graph is only build implicitly, the data structure underlying the Algorithm 1 consists of two labels for each node $j \in \{0, ..., n\}$. V_j is the cost of the shortest path from 0 to j, and P_j is the predecessor of node j. Given there are two nested loops that can iterate up to n times, the time

complexity is at most on the order of $\mathcal{O}(n^2)$. This is equivalent to CVRP since we are getting the vehicle fixed costs for each route in $\mathcal{O}(1)$ thanks to the pre-construction of the *Load2vehicle* dictionary. This time complexity can be maintain even for VFMP-*FC*, but only if the fleet is correlated. For an uncorrelated fleet, e.g. a fleet with big fuel-efficient vehicles, an additional loop would have to be added for the computation of assigning the cheapest vehicle to each feasible route. Then, the algorithm would grow to at most $\mathcal{O}(tn^2)$, with t the number of vehicle types (Prins (2009)).

Finally, after running the Algorithm 1, the optimal routes can be extracted in at most $\mathcal{O}(n)$ time, which occurs when the optimal solution consists of assigning only a single customer to each vehicle. More likely than not, this will not be the case, and the extraction of the chosen vehicles and their assigned trips will be faster.

Algorithm 1 Implementation of the exact *split* for heterogenous fleet of vehicles by @prins2009two. GT is the giant tour, a list of nodes in their visit order. d is a dictionary of customer demands, and $\mathbf{C}$ is a distance matrix between all pairs of customers and the depot.

$Q \leftarrow max\,(T_c)$
$Load2vehicle \leftarrow \{load : t_f \mid t_c \geq load \in \{0, ..., Q\}\}$
$V_0 = 0;\ V_i = \infty\ \forall i \in \{1, ..., n\}$
$P_0 = 0$
for i in $\{1, ..., n\}$ **do**
  $load = 0;\ j = i$
  while $j < n$ **and** $load \leq Q$ **do**
    $load\ +\!= d_{GT_j}$
    if $i = j$ **then**
      $cost = \mathbf{C}_{0,GT_j} + \mathbf{C}_{GT_j,0}$
    else
      $cost = cost - \mathbf{C}_{GT_{j-1},0} + \mathbf{C}_{GT_{j-1},GT_j} + \mathbf{C}_{GT_j,0}$
    end if
    if $load \leq Q$ **and** $V_{i-1} + cost + Load2vehicle_{load} < V_j$ **then**
      $V_j = V_{i-1} + cost + Load2vehicle_{load}$
      $P_j = i - 1$
    end if
    $j\ +\!= 1$
  end while
end for

Local Search

After splitting the giant tour using Algorithm 1, we could in principle stop, and deem the split to be our final solution. However, although the split is optimal given the giant tour, it may not be optimal, nor even good, for the underlying VFMP-*F*. Therefore, given the routes from the *split* algorithm, we apply two local search methods to further improve our current solution.

Intra-Route 2-Opt

Every route in our split could be seen as a TSP route on a subset of the assigned customers and the depot. Thus, to improve it, we can apply the local search techniques developed for the TSP. In our implementation we choose to apply *2-Opt*, which is a method that essentially eliminates all path crosses within a single trip. The *2-Opt* technique selects a non-consecutive pair of arcs within a trip and reverses their order. We implement a first-improvement version of *2-Opt* that performs every improving switch it finds, until no improving arc switches exist. To achieve an efficient implementation, for each potential *2-Opt* move, we calculate its cost in $\mathcal{O}(1)$ by only calculating the difference between the original route and the route after

the *2-Opt*. If this difference is negative, we accept the *2-Opt* move, and start looking for a new one. We continue this sequence of operations until no improving *2-Opt* exists. Applying this algorithm for all routes in our current solution produces a set of trips that correspond to the locally optimal trips in the *2-Opt* TSP neighborhood.

Inter-Route 2-Opt

The previous local search technique only applied *2-Opt* internally within each route. However, *2-Opt* moves can also be performed between two distinct routes. If this is to be done, we first need to decide whether we will permit a reassignment of vehicles to trips or not. Since allowing the reassignment makes the *2-Opt* significantly more involved and computationally costly, we implement and inter-route *2-Opt* without vehicle reassignment. That is, we only accept a *2-Opt* move between two arcs from two distinct routes, if the load of the newly created routes still fits inside the pre-assigned vehicles from the *split* algorithm.

Given an arc (x, y) from route 1 and an arc (u, v) from route 2, there are two ways of performing the *2-Opt* move. We can either delete the two arcs and replace them with arcs (x, v) and (u, y). Or we can instead replace them with (x, u) and (y, v). Thus for every potential *2-Opt* move we need to consider both scenarios and compute their new costs. We implement this naively by explicitly constructing the new routes and computing their new costs and loads. If the combined costs of the new routes are lower than the pre-*2-Opt* costs, and the new loads still fulfill the original vehicle capacity constraints, we perform the *2-Opt* move. Since we construct the two new routes explicitly, and every potential *2-Opt* move in this neighborhood has 2 distinct pair of routes, the computational load of determining whether a *2-Opt* move is improving is on the order of $\mathcal{O}(r_1 + r_2 + r_3 + r_4)$, where r_1 to r_4 are the lengths of the new routes. This corresponds to one for-loop for every new route.

Construction Method

Given all the ingredients above, namely the knn algorithm for constructing the giant tour, the splitting algorithm for heterogeneous fleet by Prins (2009), and the two local search techniques, we can put together our final construction method for the VFMP-*F*. The steps for constructing a good feasible solutions are:

1. Construct a giant tour using the knn algorithm
2. Split the tour into trips using the Prins *split* algorithm
3. Apply intra-route *2-Opt*
4. Apply inter-route *2-Opt*
5. Since inter-route *2-Opt* could have created new trips that are NOT optimal in their local *2-Opt* neighborhood, apply intra-route *2-Opt* again
6. Return resulting trips, fleet, and total costs

Starting from the same giant tour, the resulting solution of this construction method will always be the same. However, the method can easily be parametrized by the two previously mentioned parameters of the knn giant tour algorithm. Changing the starting node i and/or setting $k \geq 2$, provides for all intents and purposes essentially infinitely many giant tours to start from. Of course the number of giant tours is finite, however, there is so many of them, that we have no chance of enumerating them in a reasonable amount of time.

Iterated Local Search: ILS

Since our construction method can easily be parametrized, and we observe that different giant tours lead to different optimal splits, we can straightforwardly extend our construction method to an iterated local search. To do this, we introduce a new function that concatenates the current trips into a new giant route. This *concat* function just iterates over our current routes and links every last and first customer of two adjacent

trips (except for the first trip where the first visited customer remains linked to the depot). Given the new giant route obtained from the *concat* function, we can again apply the *split* algorithm, perform the local search of the new initial solution, and obtain the new locally optimal solution (in the *2-Opt* neihgborhood). We save this solution if it is the cheapest one we found so far, and we iterate in this loop until we run out of time or number of iterations we chose to perform. Defining the local search step as a sequence of Intra-Route *2-Opt* -> Inter-Route *2-Opt* -> Intra-Route *2-Opt*, the full ILS loop looks as follows:

- Construct a giant tour using the knn algorithm
- Split the tour into trips using the Prins *split* algorithm
- Apply the local search sequence
- Loop for k iterations over:
    1. Concatenate trips
    2. Shake the new giant tour by applying *2-Opt* to the entire giant tour
    3. *Split*
    4. Apply local search and go to 1.
- Return the best solution found over k iterations

A notable feature of this ILS algorithm is the shaking procedure. We have conjectured that a *good* giant tour leads to a good VFMP-F solution, therefore the shaking we perform is such that it improves the giant tour. More specifically, given a giant tour that resulted from the *concat* function, we apply the *2-Opt* local search on this giant tour as a form of shaking. The idea here is that *concat* most probably outputs a relatively *bad* giant tour. Therefore, we improve it inside of its own *2-Opt* neighborhood, and continue our iteration from there. The improved giant tour should in principle lead to improved VFMP-F solutions, hence *concat* + *2-Opt* on the giant tour should move us in the latent landscape of the original VFMP-F towards cheaper and cheaper solutions.

Multi-Start Iterated Local Search: MS-ILS

Finally, the ILS method is still dependent on the initial tour we construct using the knn algorithm. Therefore, we can further extended it by one more outer loop over different initial giant tours. This outer loop is called a multi-start and different initial giant tours can be achieved by either increasing the k parameter above 1, or by considering different starting nodes (or both).

This significantly increases the computational load, as we perform the entire ILS loop n-times, where n is the number of the starting giant tours.

Experiments and Results

To test the effects of the two parameters for the giant tour construction, and the different methods, namely construction method, ILS, and MS-ILS, we perform 7 different experiments across 12 test instances. The test instances increase in the number of customers, and originally come from Christofides and Eilon (1969). For every instance we compare our results to the best known solution, and present the optimality gap alongside the time it took to reach the final solution.

Experiment 1

We start our experimentation with a baseline construction method, where $k = 1$ and the depot is the starting node. Therefore, the acquired solutions displayed in Table 1 are deterministic. Across the 12 instances, the best achieved solution has an optimality gap of 6%, while the worst one 13.8%. All solutions are realized

within 1 second, except for the two largest instances with 100 customers. The results in Table 1 will serve as a point of comparison for the subsequent experiments.

Table 1: Summary results of Experiment1. Average optimality gap 0.078

instance	optimal	cost	gap	time
03°E051-05e	961.03	1018.73	0.060	0.060
04°E051-05e	6437.33	6943.65	0.079	0.037
05°E051-05e	1007.05	1072.71	0.065	0.043
06°E051-05e	6516.47	7415.26	0.138	0.042
13°E076-08s	2408.62	2548.27	0.058	0.379
14°E076-08s	9119.28	9717.26	0.066	0.366
15°E051-05e	2586.37	2855.10	0.104	0.352
16°E051-05e	2741.50	2949.11	0.076	0.330
17°E076-10e	1749.50	1889.90	0.080	0.796
18°E076-10e	2381.43	2512.28	0.055	0.882
19°E101-08e	8675.16	9339.80	0.077	1.463
20°E101-08e	4086.76	4421.98	0.082	1.503

Experiment 2

Following the deterministic Experiment 1, we set $k = 2$ and look at what happens when the construction method is allowed to be partly stochastic. Table 2 shows the results of 20 independent runs of the construction method with $k = 2$. We can see that the optimality gap drops on average across all instances from 7.8% in Experiment 1, to 5.6% in Experiment 2 (here we make the comparison to the best solutions found in Experiment 2). We also notice, that comparing the optimality gap in Experiment 1 with the mean gap in Experiment 2, the former outperforms the latter. If the assertion that the giant tours constructed with $k = 1$ are on average better than their stochastic version with $k = 2$ holds, we find some evidence for the conjecture that better giant tours lead to better VFMP-F solutions. This is of course only true in a certain *goodness* interval as we have previously said that in general the optimal giant tour does not necessarily lead to the optimal VRP solution. In fact, if we look at the best solutions found over the 20 re-runs, we find that the minimal optimality gaps for Experiment 2 are in all but 2 instances (18°E076-10e & 19°E101-08e) better than the solutions found in Experiment 1. As expected, the total running times have increased, however, we can see that the running times obtained in Experiment 1 were not representative of the average running times. In fact the average running times per instance seem to range from 0.017s in instance 03°E051-05e to 0.541s in instance 20°E101-08e (averaged over 20 runs which is still not a representative sample).

Table 2: Summary results of Experiment2. Average minimum optimality gap 0.056

instance	optimal	mean_cost	min_cost	mean_gap	min_gap	total_time
03°E051-05e	961.03	1043.80	990.63	0.086	0.031	0.342
04°E051-05e	6437.33	7208.66	6929.01	0.120	0.076	0.346
05°E051-05e	1007.05	1094.06	1031.60	0.086	0.024	0.395
06°E051-05e	6516.47	7451.62	6992.53	0.144	0.073	0.280
13°E076-08s	2408.62	2577.87	2490.82	0.070	0.034	2.468
14°E076-08s	9119.28	9837.92	9624.12	0.079	0.055	1.786
15°E051-05e	2586.37	2839.95	2732.05	0.098	0.056	1.810
16°E051-05e	2741.50	2983.88	2856.70	0.088	0.042	1.915
17°E076-10e	1749.50	1960.44	1857.22	0.121	0.062	4.834
18°E076-10e	2381.43	2622.13	2553.09	0.101	0.072	5.814

instance	optimal	mean_cost	min_cost	mean_gap	min_gap	total_time
19°E101-08e	8675.16	9484.37	9371.84	0.093	0.080	10.584
20°E101-08e	4086.76	4443.58	4369.78	0.087	0.069	10.816

Experiment 3

The 3rd experiment further probes the effect of the parameter k. Everything is kept the same as in Experiment 2, just $k = 3$ this time around. As we can see in Table 3, both the average and the minimal optimality gaps increase in comparison to both Experiment 1 and 2. This provides further evidence for the statement that *good* giant tours lead to *good* VFMP-F solutions. Given the results from this experiment, we conclude it is more beneficial to achieve stochasticity in the giant tour construction with $k = 2$, than $k = 3$. We extrapolate that this holds for larger values of k as well, since the larger the k the more random the giant tour.

Table 3: Summary results of Experiment3. Average minimum optimality gap 0.078

instance	optimal	mean_cost	min_cost	mean_gap	min_gap	total_time
03°E051-05e	961.03	1066.21	1009.25	0.109	0.050	0.498
04°E051-05e	6437.33	7337.65	6926.24	0.140	0.076	0.344
05°E051-05e	1007.05	1127.62	1078.78	0.120	0.071	0.377
06°E051-05e	6516.47	7512.50	7072.02	0.153	0.085	0.400
13°E076-08s	2408.62	2636.90	2551.36	0.095	0.059	2.571
14°E076-08s	9119.28	9809.80	9686.99	0.076	0.062	2.226
15°E051-05e	2586.37	2880.29	2796.23	0.114	0.081	2.453
16°E051-05e	2741.50	3038.10	2979.79	0.108	0.087	1.941
17°E076-10e	1749.50	2028.40	1934.07	0.159	0.105	7.117
18°E076-10e	2381.43	2685.13	2621.63	0.128	0.101	7.590
19°E101-08e	8675.16	9526.55	9388.35	0.098	0.082	12.751
20°E101-08e	4086.76	4502.47	4406.41	0.102	0.078	12.223

Experiment 4

In the 4th experiment we probe the effects of varying the starting node parameter. We set the value of k to 1, in order to isolate the effect from stochasticity that results from $k \geq 2$. Table 4 shows the average and the best results from constructing the giant tour starting at every single node in the instance. Averaging the best results, we achieve the smallest average optimality gap of 5.1% thus far. In comparison to the Experiment 1, we can see that in all instances there has been at least one customer who when used as the starting node, resulted in a more optimal giant tour for our objective. This is the case because all the minimal gaps in Table 4 are strictly smaller than the optimality gaps in Table 1. Furthermore, since we are iterating over all the nodes in each instance, the for-loop in the instance with 100 customers is longer that the same for-loop in the 20 customer instance. Therefore, as expected we observe increased running times across all instances, as well as more significant growth in the running time with respect to the size of the instance.

Table 4: Summary results of Experiment4. Average minimum optimality gap 0.051

instance	optimal	mean_cost	min_cost	mean_gap	min_gap	total_time
03°E051-05e	961.03	1020.32	1003.98	0.062	0.045	0.313
04°E051-05e	6437.33	7317.87	6924.24	0.137	0.076	0.234

instance	optimal	mean_cost	min_cost	mean_gap	min_gap	total_time
05°E051-05e	1007.05	1080.29	1045.87	0.073	0.039	0.281
06°E051-05e	6516.47	7249.35	7006.22	0.112	0.075	0.338
13°E076-08s	2408.62	2532.77	2465.60	0.052	0.024	5.820
14°E076-08s	9119.28	9666.80	9599.87	0.060	0.053	6.388
15°E051-05e	2586.37	2801.29	2675.32	0.083	0.034	6.619
16°E051-05e	2741.50	2950.51	2884.40	0.076	0.052	6.287
17°E076-10e	1749.50	1893.34	1837.33	0.082	0.050	20.523
18°E076-10e	2381.43	2530.31	2475.21	0.063	0.039	24.697
19°E101-08e	8675.16	9499.03	9250.70	0.095	0.066	54.565
20°E101-08e	4086.76	4407.54	4333.09	0.078	0.060	52.900

Experiment 5

In Experiment 5, we move away from probing the effects of the two parameters of the knn construction algorithm, to looking at the results of the Iterated Local Search. As described in the Methods section we perform the ILS loop with 20 iterations and the initial giant tour parameters set to $k = 1$ *starting_node* = *depot*. Table 5 shows the costs, optimality gaps, and running times of the best found solutions. We note that the average optimality gap is the lowest thus far at a mere 4%, and for two instances (03°E051-05e & 13°E076-08s) we have found sub 1% solutions. Moreover, the running times are notably shorter than those in Experiment 4, which makes the ILS with 20 iterations a superior method when compared to iterating over all nodes in our instances and applying only a single iteration of local search.

Table 5: Summary results of Experiment5. Average optimality gap 0.04

instance	optimal	cost	gap	time
03°E051-05e	961.03	978.93	0.019	0.394
04°E051-05e	6437.33	6943.65	0.079	0.359
05°E051-05e	1007.05	1010.12	0.003	0.379
06°E051-05e	6516.47	7365.98	0.130	0.446
13°E076-08s	2408.62	2424.63	0.007	3.198
14°E076-08s	9119.28	9598.71	0.053	3.456
15°E051-05e	2586.37	2698.61	0.043	3.270
16°E051-05e	2741.50	2831.02	0.033	3.086
17°E076-10e	1749.50	1796.63	0.027	6.489
18°E076-10e	2381.43	2431.96	0.021	9.130
19°E101-08e	8675.16	9017.40	0.039	18.813
20°E101-08e	4086.76	4207.84	0.030	20.188

Experiment 6

Finding very promising results in Experiment 5, we perform MS-ILS in Experiment 6 with 20 iterations per starting solution and 15 starting solutions. Since initial giant tours constructed with $k = 2$ have yielded better VFMP-F solutions than those with $k = 3$, we achieve different starting giant tours with setting $k = 2$. Table 6 shows the results of MS-ILS, which can be seen as the best performing method in terms of average optimality gaps thus far. The average optimality gap is only 2.5% and we have found sub 1% solutions in 4 cases. However, the computational load can clearly be visible as running times increased significantly. This could partially be remedied by a parallel implementation of our method, since MS-ILS is in principle easily parallelizable.

Table 6: Summary results of Experiment6. Average optimality gap 0.025

instance	optimal	cost	gap	time
03°E051-05e	961.03	961.03	0.000	6.465
04°E051-05e	6437.33	6899.08	0.072	4.445
05°E051-05e	1007.05	1010.12	0.003	6.806
06°E051-05e	6516.47	6997.96	0.074	6.094
13°E076-08s	2408.62	2422.00	0.006	48.648
14°E076-08s	9119.28	9120.35	0.000	52.656
15°E051-05e	2586.37	2639.39	0.020	52.047
16°E051-05e	2741.50	2798.17	0.021	41.721
17°E076-10e	1749.50	1778.11	0.016	98.988
18°E076-10e	2381.43	2428.03	0.020	137.575
19°E101-08e	8675.16	9034.04	0.041	295.492
20°E101-08e	4086.76	4187.68	0.025	273.150

Experiment 7

Finally, to test the trade-off between multiple starts and the number of iterations in each start, we carry out MS-ILS with 25 different starting solutions, and only 5 iterations of ILS per solution. Table 7 shows the results of this experiment. The average optimality gap is slightly worse than in Experiment 6, 3%, however, it is still the second best achieved average optimality gap. More crucially, the running times are roughly half of those in Experiment 6, which suggests that trading multiple starts for number of iterations in each start may be a beneficial tactic for achieving very good solutions in faster time.

Table 7: Summary results of Experiment7. Average optimality gap 0.03

instance	optimal	cost	gap	time
03°E051-05e	961.03	961.03	0.000	3.265
04°E051-05e	6437.33	6892.14	0.071	2.642
05°E051-05e	1007.05	1010.12	0.003	3.003
06°E051-05e	6516.47	6995.71	0.074	2.629
13°E076-08s	2408.62	2424.45	0.007	24.599
14°E076-08s	9119.28	9172.41	0.006	21.703
15°E051-05e	2586.37	2670.97	0.033	23.317
16°E051-05e	2741.50	2809.23	0.025	21.812
17°E076-10e	1749.50	1798.58	0.028	47.927
18°E076-10e	2381.43	2447.17	0.028	63.090
19°E101-08e	8675.16	9181.77	0.058	144.688
20°E101-08e	4086.76	4184.61	0.024	126.372

Conclusion

In conclusion, we have developed and tested three increasingly more complex methods for solving VFMP-F. We find that MS-ILS achieves the most optimal results, however, the running times are much higher than the running times for the construction method or plain ILS. This could partially be remedied by implementing MS-ILS in a parallel fashion, which has not been done in our implementation. Moreover, a number of further extensions to our methods could still be implemented. For example, in MS-ILS, the ILS could be terminated

when no improving solutions are found in n subsequent iterations. This could in principle save time, and only explore those initial solutions that lie in the valleys of our latent landscape of all feasible solutions. Alternatively, more varied experiments on a larger number of instances could be carried out to determine the most optimal combination of the parameters of the knn algorithm, number of iterations, and the number of multi-starts.

References

Beasley, John E. 1983. "Route First-Cluster Second Methods for Vehicle Routing." *Omega* 11 (4): 403–8.

Christofides, Nicos, and Samuel Eilon. 1969. "An Algorithm for the Vehicle-Dispatching Problem." *Journal of the Operational Research Society* 20 (3): 309–18.

Golden, Bruce, Arjang Assad, Larry Levy, and Filip Gheysens. 1984. "The Fleet Size and Mix Vehicle Routing Problem." *Computers & Operations Research* 11 (1): 49–66.

Prins, Christian. 2009. "Two Memetic Algorithms for Heterogeneous Fleet Vehicle Routing Problems." *Engineering Applications of Artificial Intelligence* 22 (6): 916–28.