A Multi-start SCI Algorithm to the CVRPTW

H.E. Eisen R.K. Dijkstra F.D. Brunet de Rochebrune

Group 33, Combinatorial Optimisation, Vrije Universiteit Amsterdam,
7 May 2020

1 Introduction and Proof of NP-Completeness

With this paper we aim to find an approach that meets the criteria as given per the assignment; a qualitative solution to the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). A detailed description of this CVRPTW can be found here. The division of responsibilities of each team member can be found in Table 1.

The CVRPTW is a member of the family VRPs, it is a member of this class since it contains the Vehicle Routing Problem, with additionally limited capacity for each of the vehicles and the restriction of deliveries and pickups in certain time windows.

For a search problem to be defined as NP-Complete the following two conditions must be met: 1) It must be in NP. 2) Every other problem in NP is reducible to it [4]. NP is defined as the class of all search algorithms, P is defined as the class of search problems that can be solved in polynomial time. Firstly, it is easily deduced that CVRPTW belongs to NP; we can inspect the correctness of a solution in polynomial time by going through the solution and checking whether constraints have been violated. Later, it is proven that finding a solution cannot be done in polynomial time. As previously stated, CVRPTW is a member of the VRP-class. In order to prove the NP-completeness of the CVRPTW class we can use several reductions. The CVRPTW can be transformed into CVRP by 'relaxation' of time windows, likewise CVRP can be transformed into VRP by 'relaxation' of vehicle capacities. We now remain with the VRP. The Vehicle Routing Problem can be seen as a generalisation of the Travelling Salesman Problem [9]. We can reduce in the following way: If k vehicles are used for the VRP and the depot is located at node 0, we can add $k-1$ artificial depots at the location of the original depot (there are thus k depots, these depots are not connected). The VRP is transformed into MTSP (Multiple Travelling Salesman Problem), which is consequently transformed into a TSP. [3] In order to find a solution to the VRP, we must find a solution for the TSP on this adjusted graph.

For the TSP we must note two variants exist, being 1) TSP: find shortest path visiting all nodes exactly once and returns at the starting node. 2) decision-TSP: given k, search whether the graph has a tour of length $\leq k$ [11]. The TSP problem is known to be NP-hard, and the decision variant is known to be NP-Complete (which can be proved via reduction from Rudrata Cycle, which is also known to be NP-Complete). As we know the VRP is a generalisation of the TSP, and TSPs are NP-Complete / NP-hard. We can conclude VRPs are NP-Complete / NP-hard as well. As the CVRPTW is a generalisation of CVRP, CVRP is a generalisation of the VRP [6] and the VRP is NP-Complete, we can conclude that the search/decision problem for CVRPTW is an NP-complete problem. Finding the optimal solution to CVRPTW is therefore considered to be a NP-hard problem and searching a solution therefore requires careful consideration.

2 Literature and Methods

2.1 SWEEP

In the past, the cluster first route second method was a commonly used and efficient method for the VRP. This approach divides the problem into a clustering problem and a routing problem. The latter can be done using a wide variety of algorithms, including solving it as a Traveling Salesman Problem. The goal of this problem is to find the shortest route through all nodes, visiting each node

once. The clustering can be done following several methods, of which we chose to try implementing an algorithm inspired by the SWEEP algorithm.

The clustering can for example be done following the traditional sweep. Starting from the depot, a line originating in the depot is drawn between the nodes that are furthest away from each other [1], after which this line is moved until another node is found. The cluster is extended with the next node if this fits the capacity. This process is repeated until capacity feasible clusters are created and all nodes are visited once. The largest angle (in polar coordinates) θ_i between the depot and node i (location of a request) determines the boundaries of a cluster. The 'sweeping' can be done counterclockwise (backward sweep) and clockwise (forward sweep) and should be done based on performance. There are also other options for creating clusters, such as the simple sweep, window-wise sweep, and corrective sweep [7]. These variants differ from the traditional sweep as the nodes are added under different constraints (such as time windows).

Our implementation of the SWEEP algorithm was to create clusters based on k-means clustering algorithm (for pickups and deliveries separately). For these clusters we constructed routes by means of TSPs, while consequently checking for capacity constraints. Our clustering implementation deviated from the original SWEEP by differing initialisation of centroids per run, representing the sweeping aspect. With this implementation, there were vehicles going out for delivery and vehicles going out for pickup, which was not that effective. We therefore used the same implementation such that one vehicle could carry out both pickups and deliveries. For this situation we encountered several problems regarding choices on number of clusters, as well as problems on route initialisation within clusters, since the TSP did not construct paths coherent with capacity constraints, yielding inefficient or infeasible routes. Furthermore this approach forced us to do multiple runs (as to resemble the sweeping part) and had relatively long run-times in total as solving TSPs and clustering was computationally very demanding. Lastly, clustering this way often yielded imbalanced cluster sizes. We concluded we preferred a heuristic which would create routes based on the capacity of vehicles at different points in routes, such that pickups and deliveries could easily be combined, yet being computationally fast and not violating distance constraints. Although we saw the usefulness of the cluster first route second approach, we preferred heuristics capable of grouping sets of requests based on proximity and feasibility.

2.2 Cheapest Insertion

Early heuristics were implemented in solving VRPs, with one of the algorithms being (Sequential) Cheapest Insertion (SCI) . This algorithm works as follows [5, 10]:

1. A pivot is chosen based on the capacity and distance of a request's location relative to the depot (Formula 2).

2. For this pivot the algorithm considers all nodes which are still reachable given capacity and distance constraints. For each possible node the extramileage is calculated according to Formula 1 and illustrated by Figure 1. If a certain node violates distance or capacity constraints for a given route, the extra mileage is set to infinity.

3. The node to be inserted into the route is selected based on minimum extramileage.

4. If no nodes are reachable given distance and capacity constraints, a route will be ended and a new pivot is chosen based on Formula 2 and a vehicle is added. For this new pivot, extramileage calculations are repeated for the unrouted nodes.

5. All previous steps are repeated until all nodes are routed.

$$\mu_{h,i} = c_{i,h} + c_{h,i} - c_{i,\sigma_i} \tag{1}$$

In Formula 1, $\mu_{h,i}$ represents the extramileage to insert node h in route i, j. This

consists of the difference in distance between route $.. - i - j - ..$ and $.. - i - h - j - ...$

$$d_i = \frac{q_i}{q_{max}} + \frac{c_i}{c_{max}} \tag{2}$$

In Formula 2, d_i represents the value of the (possible) pivot node i, α and β the weights for the demand (q) and distance from depot to location i (c) respectively. As we believe it is also important to take into account if a request is a delivery or pickup, we add in a term for delivery with corresponding weight "pickup factor" (p). A pivot is from now on determined by choosing the maximum of:

$$d_i = \frac{q_i}{q_{max}} + \frac{c_i}{c_{max}} + p\rho \tag{3}$$

$$\text{where } \rho = \begin{cases} 0 & \text{if request is delivery} \\ 1 & \text{if request is pickup} \end{cases}$$

This way, a pickup is more likely to become a pivot and a vehicle is more likely to pick up some goods first that it can possibly deliver later. Moreover, we decided that we would have better results if the weights would differ per instance type. For example if there are high tool costs, it is more important to find routes that execute a pickup before a drop off (which could lead to a vehicle driving longer routes), whereas if there are high distance costs, it is more important that a vehicle chooses the 'most difficult' pivots: pivots that are far from the depot, as to eliminate them from being chosen in later routes inefficiently.

In order to determine what the weights should be for each instance type, we conducted a trial in which we tried all possible combinations of weights for each instance and recorded these weights together with the costs of that instance. We then looked further into the ten best combinations of all four occurrences of each instance type that yielded the lowest costs. The results of this investigation are presented in the Figures 2 – 6. From these figures, we concluded that the parameters as presented in Table 2 yielded better results than keeping the same parameters for each instance type.

Implementing SCI decreased the runtime of our program significantly and also eradicated the SWEEP's re-clustering aspect, while constructing seemingly qualitative routes. A drawback of Sequential Cheapest Insertion is that it is a greedy algorithm, it does not incorporate the effect of choices it makes. According to Alssager et al. [2], Multi-start helps combatting the greediness of Sequential Cheapest Insertion. Thus to find better solutions, we also take a look into implementing Multi-start in our algorithm.

2.3 Multi-Start

Multi-start algorithms (MS) operate according to the philosophy of: "Different executions of the algorithm generally lead to different solutions" [12]. We can therefore run multiple randomised versions of the algorithm until the time budget has been reached and store the best result. For a multi-start algorithm to be implemented the algorithm must be available in a randomised/parametric version. To transform the SCI algorithm to a randomised SCI algorithm, randomness can be incorporated in many ways, such as changing pivot selection, defining route cut-off criteria or via extra mileage calculations. The idea of Multi-start is described by the following [5]:

1. Define a stopping criterion (in our case a runtime).

2. Run and store the solution (costs). This is automatically the best found solution as it is the only solution.

3. While the stopping criterion is not met yet (hence if time allows), keep looking for a better solution by running a randomised version of the algorithm. If a found solution is better than the old best solution found, update this accordingly.

4. If the stopping criterion is met, stop executing the algorithm and return the best found solution.

3 Our Implementation

3.1 Scheduling

Our final implementation consists of a schedule first route second approach, inspired by the cluster first route second approach. The scheduling approach has remained the same throughout our implementation. Scheduling is done naively, in a way that enough resources are available at the depot such that one vehicle can carry out one request, after which it would return to the depot [8]. This yields a feasible schedule that we can then further elaborate upon, by means of efficient routing. The scheduling is tackled by grouping the request into four different groups:

1. High priority requests. These requests are requests that can only be delivered on a certain day rather than within a time window.

2. Short delivery timeframe requests. These are the requests that can only be delivered on two days.

3. Short request duration requests. These are the requests that only need the tool for one day.

4. Other requests. These are the requests that have a larger time window and/or need the tool for longer.

By following this order in executing the requests, we ensure that the tools that need to be delivered on a certain day (and cannot be moved to another day) are scheduled that day. The other requests are also scheduled, for as long as there are still tools available at the depot. Is this not the case, then the delivery day for those requests is moved a day. This approach seemed to work well, however it had some flaws. This was due to the fact that sometimes different tools were overused due to mis-scheduling, resulting in a tool deficit on some days. This problem was solved by retrying the scheduling part more efficiently, by ensuring an even spread of requests with high tool usage over the days, yielding feasible solutions for all instances.

3.2 Routing

Regarding the final implementation of the routing, several options have been explored and implemented, as an extension to Sequential Cheapest Insertion (SCI). The first being clustering of requests, secondly pivot selection, lastly, the transformation to a multi-start algorithm.

3.2.1 Clustering

We examined the use of implementing clustering algorithms on top of Sequential Cheapest Insertion. What we noted however, after implementation, was that cluster sizes were imbalanced, yielding very short routes for some clusters. Moreover we noted that for the big clusters, vehicle routes eventually became short; a vehicle would be sent back after the final request was finished (as there were no remaining requests in that cluster), even though the vehicle's capacity or maximum distance was not reached yet. We therefore decided that clustering was not an efficient way to proceed and therefore to omit the clustering algorithm from our implementation. Although cluster sizes may become balanced by reducing the number of clusters, it still does not solve the latter posed problem. Therefore, no clustering is performed in our final implementation.

3.2.2 Pivot Selection

As previously described we have elaborated on the basic pivot choice algorithm as described in Formula 2. At first this pivot selection formula yielded some good results, however, after careful investigation, we have decided to adapt pivot selection as described in Formula 3, based on previous argumentation, to prioritise pickups at the beginning of a route. As instance types have different cost structures, we should adapt pivot selection accordingly by means of adapting weights (α, β, p). For our implementation we have selected these weights based on the scheme proposed in Section 2, these final weights have been implemented in the final version of our program.

3.2.3 Multi-start

We transformed our final implementation to a randomised/parametric version by means of

randomisation of scheduling, and by randomisation of pivot selection. As can be seen in Figures 2 – 6, most weights for a specific instance type are not clearly better than others, yielding an opportunity for randomisation. To prevent randomness from completely taking over the route construction, we restrict randomness in pivot selection by constructing reasonably small intervals (for each instance type, see 3, based on initial pivot weight research in Figures 2 – 6), from which weights are drawn uniformly. For each run random pivot weights will be incorporated. Implementation of MS yields the ability to explore a wider variety of solutions.

In conclusion, our final routing implementation consists of a Multi-start Sequential Cheapest Insertion implementation, extended with amended pivot selection.

3.3 Runtime Analysis

Regarding the runtime of our final implementation, we can separate it in two parts. A single run and the MS. For the single run we must distinguish the following two aspects, namely scheduling and SCI. For the time complexity of scheduling we can observe that the scheduling algorithm iterates over all d days, for each day considering all n requests to check whether they can be delivered or not, pickup dates are consequently automatically determined. This scheduling method yields a time complexity of $\mathcal{O}(dn)$. As for the SCI part (with a given schedule), a request to be handled can either be a pickup or delivery, hence in total $2n$ 'requests' must be fulfilled in d days. As a request cannot be delivered and picked up on one day, only at most n requests can be handled on a given day. So therefore, given a schedule of n requests to be completed on a given day we obtain the following runtime analysis. At first out of n requests a pivot must be chosen: $\mathcal{O}(n)$. Given a pivot, extra mileages are calculated for at most $n-1$ requests, subsequently minimum extra mileage is determined and a node is inserted into a route, yielding a total complexity of $\mathcal{O}(n)$. After a request is inserted, extra mileages are calculated again for the $n-2$ customers, again with $\mathcal{O}(n)$; and so forth. The maximum number of extra mileage calculations, for a pivot, can therefore be denoted by $\sum_{i=1}^{n-1} n-i$, having time complexity $\mathcal{O}(n^2)$. To summarise, as at most n pivots can be determined and for each pivot the route construction's time complexity equals $\mathcal{O}(n^2)$, the total time complexity of SCI thus equals $\mathcal{O}(n^3)$ [5].

As for the Multi-start part, runtime is constrained by total budget (in our case 15 seconds), within this time several calls to SCI can be made, until the budget is depleted. This Multi-start part has therefore no major implications on the time complexity, beside constant time required for writing and validating solutions.

4 Results and Discussion

With the twenty available instances we were able to deduce insights and compare results pretty well. The instance files are divided into five different types, with each their own cost structure. Our program responds differently to each type in its best way, but of course no solution is optimal, which means there are always some shortcomings.

We optimised our schedule first route second approach via implementation of Sequential Cheapest Insertion (SCI) to create feasible routes and not immediately return to the depot. With sequential cheapest insertion we look at all possibilities to extend our route as far as possible. This way we have the advantage of creating long routes, taking multiple requests into one vehicle. This results in hypothetically less vehicles being used and less tools used; consequently lower costs. However, longer routes also may implicate higher total distance costs. Moreover, effective combination of pickups and deliveries of the same tool type is not guaranteed.

First, if distance costs for an instance type are relatively high (as compared to other costs) we should be aware of the termination of SCI. SCI does not take total distance (travelled) so far into account, and has therefore no way of effectively limiting total distance travelled. The inability of minimising distance travelled may occur at early and late constructed routes. For the early constructed routes, the SCI will find capacity-feasible nodes, while not checking the distance

required to travel to these. SCI therefore has no way to cut-off route construction if distance costs are substantial. Looking closer at Route 6 in Figure 7, it can be noted that the nodes chosen after the pivot are not in the (near) proximity of the pivot itself. Again, SCI operates greedily; it has no way of effectively minimising distance when a few nodes remain, and to decide to construct new routes if distances to be travelled become too large. Therefore, when few nodes remain and routes are created, they either have long distance travelled, or contain few nodes, as is also visible in Route 8 in Figure 7. Thus, the (last) constructed routes can be inefficient distance-wise or capacity-wise.

Moreover, we should be aware of the occurrence of 'stray nodes' near the end of the algorithm. An example of this can be seen in Figure 8. Routes 5 and 6 both consist of only one request, this may be due to the greedy route construction. SCI does not take into account possible routings after an extra mileage insertion, therefore greedily chosen extramileages are not necessarily the most efficient choices. This leaves other nodes behind (which might be better candidates capacity wise) possibly to be routed by another vehicle. What we have noticed frequently in route constructions, is that these 'better' candidates remain unserviced due to greediness of SCI, leaving them to be inefficiently routed for the last few routes. This inefficiency may backfire in distance to be travelled, unnecessarily adding vehicles or by letting opportunities pass to more efficiently combine pickups and deliveries. Routes 5 and 6 in Figure 7 are an example of these 'one-request routes.'

We believe that major improvements to our algorithm can be made, for example with carefully combining pickups and deliveries. As for our current implementation no careful consideration on this matter is done. Another approach would be to tackle scheduling by means of creating pickup and delivery pairs, by for example formulating it as an ILP. This approach could decrease the tool usage costs considerably, as previously picked up tools can be delivered to other customers; in our current implementation there is no way of guaranteeing this behaviour, as we solely look at the general capacity of a vehicle and not its contents.

Other improvements can be made by changing the way pivots are chosen by adding a factor that takes the distance between other pivots into account. Another way is to adjust the chosen pivots based on initially created routes, as to select these latter called 'stray-nodes' or 'hard requests' as pivots. This way we overcome the drawbacks of adding vehicles unnecessarily. Moreover, we could implement a distance check, cutting off routes prematurely to prevent routes from becoming too long or inefficient, weighing off the benefit of long routes against unnecessarily adding vehicles (to serve stray nodes).

5 Conclusion

This research was performed to approach a solution to the CVRPTW with reasonable speed, based on previous research. We went from a simple naive implementation to a full-fledged algorithm while maintaining relatively quick runtimes. What struck us the most was the running time after implementing the Sequential Cheapest Insertion algorithm, yet maintaining good solution quality. SCI outperformed our SWEEP-inspired k-means clustering algorithm, not only in performance, but especially in runtime. We can conclude that our research goal was reached, and an approach to solving this CVRPTW was found. Due to the nature of the problem (its NP-completeness), however, there are always improvements to be made, including to our implementation.

Examples of improvements, as mentioned in Section 4, are changing the way pivots are chosen, changing the naive scheduling implementation, e.g. using ILPs. Moreover, the current algorithm is greedy, resulting in varying solution qualities. We acknowledge that there is room for improvement regarding the implications of greediness on our solution quality. Due to time constraints these matters could not be further researched upon (to solve them), stressing the need for further research. However, we are still satisfied with the solution quality and its runtime.

References

[1] MAH Akhand, Zahrul Jannat Peya, and Kazuyuki Murase. "Capacitated vehicle routing problem solving using adaptive sweep and velocity tentative PSO". In: *International Journal of Advanced Computer Science and Applications* 8.12 (2017), pp. 288–295. DOI: 10.14569/IJACSA.2017.081237.

[2] M. Alssager, Z.A. Othman, and M. Ayob. "Cheapest Insertion Constructive Heuristic based on Two Combination Seed Customer Criterion for the Capacitated Vehicle Routing Problem". In: *International Journal on Advanced Science, Engineering and Information Technology* 7.1 (2017), pp. 207–214. URL: https://www.researchgate.net/profile/Zulaiha_Ali_Othman/publication/314084771_Cheapest_Insertion_Constructive_Heuristic_based_on_Two_Combination_Seed_Customer_Criterion_for_the_Capacitated_Vehicle_Routing_Problem/links/58da6eb545851578dfb8247d/Cheapest-Insertion-Constructive-Heuristic-based-on-Two-Combination-Seed-Customer-Criterion-for-the-Capacitated-Vehicle-Routing-Problem.pdf.

[3] T. Bektas. "The multiple traveling salesman problem: an overview of formulations and solution procedures". In: *Omega* 34(3) (2006). DOI: 10.1016/j.omega.2004.10.004.

[4] S.A. Cook. "The complexity of theorem proving procedures". In: *Proceedings, Third Annual ACM Symposium on the Theory of Computing, ACM, New York* (1971).

[5] J. Gromicho and D. Vigo. *Transport and Distribution Planning*. 2018.

[6] A.H. El Hassani, L. Bouhafs, and A. Koukam. "A Hybrid Ant Colony System Approach for the Capacitated Vehicle Routing Problem and the Capacitated Vehicle Routing Problem with Time Windows". In: *Systems and Transportation Laboratory* (2008). DOI: 10.5772/5640.

[7] C. Hertrich, P. Hungerländer, and C. Truden. "Sweep Algorithms for the Capacitated Vehicle Routing Problem with Structured Time Windows". In: *Operations Research Proceedings 2018* (2019), pp. 127–133. URL: https://arxiv.org/abs/1901.02771v1.

[8] A. Kheiri et al. "Tackling a VRP Challenge to redistribute scarce equipment within time windows using metaheuristic algorithms". In: *EURO J Transp Logist* 8 (2019). DOI: 10.1007/s13676-019-00143-8.

[9] G. Laporte and Y. Nobert. "Exact algorithms for the vehicle routing problem". In: *North-Holland Mathematics Studies* 132 (1987), pp. 147–184. DOI: 10.1016/S0304-0208(08)73235-3.

[10] R.H. Mole and S.R. Jameson. "A sequential route-building algorithm employing a generalised savings criterion". In: *Journal of the Operational Research Society* 27.2 (1976), pp. 503–511. DOI: 10.1057/jors.1976.95.

[11] M. Prates et al. "Learning to Solve NP-Complete Problems: A Graph Neural Network for Decision TSP". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019). DOI: 10.1609/aaai.v33i01.33014731.

[12] D. Vigo and J. Gromicho. "Routing". University Lecture. 2020.

Appendices

A Work Division

Name	Role + Responsibilities
H.E. Eisen	Implemented naive scheduling + routing (one vehicle one request). Implementation of CI (partially with Renze). Basis for 'Our Implementation'. Literature + methods. Major part in converting the (Google drive) document to a Latex document. Helped Florent with Results, Discussion and Conclusion.
R.K. Dijkstra	Proof NP-Completeness. Major part on 'Our Implementation'. Implemented SWEEP-based implementation of routing. Collaborated with Heleen on Cheapest Insertion. Hyperparameter selection of CI. Helped Florent with Multi-start Cheapest Insertion. Runtime Analysis algorithm. Construction of Route plots. Helped Florent with Results & Discussion.
F.D. Brunet de Rochebrune	Code performance optimisation. Orchestrated server environment. Implemented Multi Start and final runtime. Results, discussion and conclusion.

Table 1: Contribution Overview

B Extra Mileage Insertion

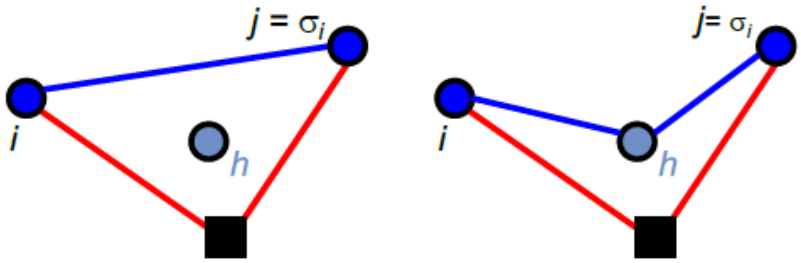

Figure 1: Visualisation of Extramileage [5]

C Weights Selection for Pivot Selection

Instance Type	β	α	p
Type 1	0.1	0.9	0.35
Type 2	0.1	0.8	0.5
Type 3	0.1	0.95	0.0
Type 4	0.03	0.95	0.1
Type 5	0.05	0.95	0.3

Table 2: Optimal Weights For Each Instance Type

Instance Type	β	α	p
Type 1	0.05 – 0.15	0.85 – 0.95	0.25 – 0.50
Type 2	0.00 – 0.30	0.75 – 1.00	0.30 – 1.00
Type 3	0.05 – 0.55	0.65 – 1.00	0.00 – 0.10
Type 4	0.00 – 0.20	0.75 – 1.00	0.00 - 0.25
Type 5	0.00 - 0.15	0.85 – 1.00	0.15 – 0.45
Otherwise	0.00 – 1.00	0.00 – 1.00	0.00 – 1.00

Table 3: Chosen Weight Intervals For Each Instance Type

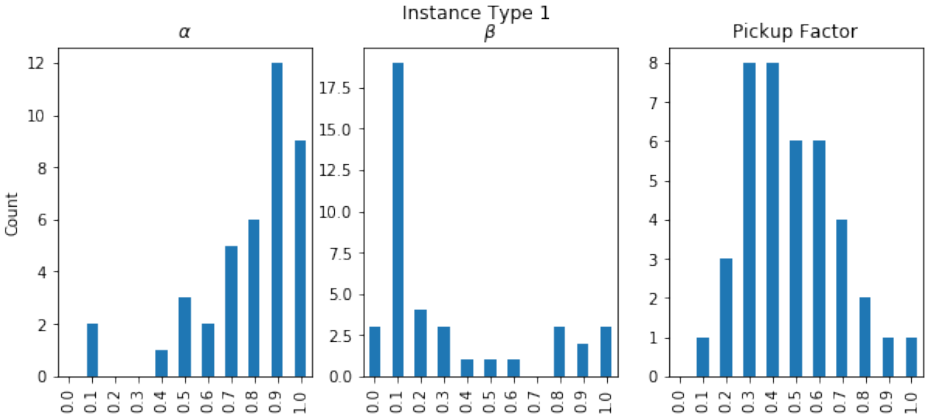

Figure 2: Occurrence of Weight in Instance Type 1

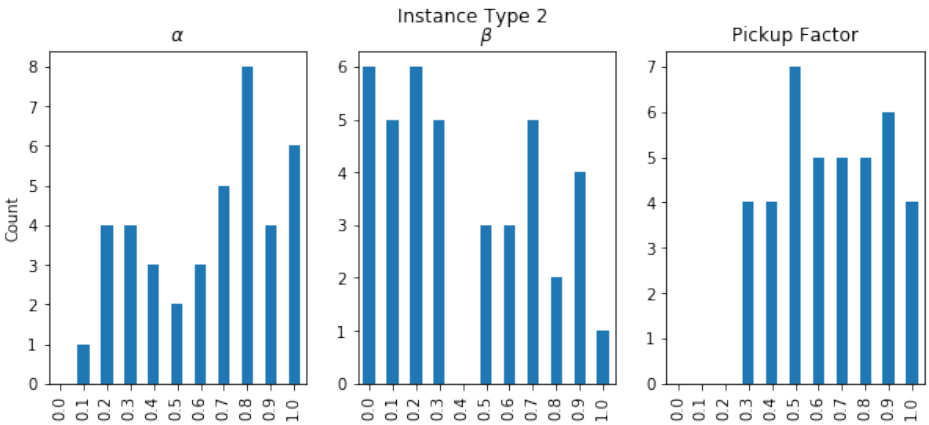

Figure 3: Occurrence of Weight of Instance Type 2

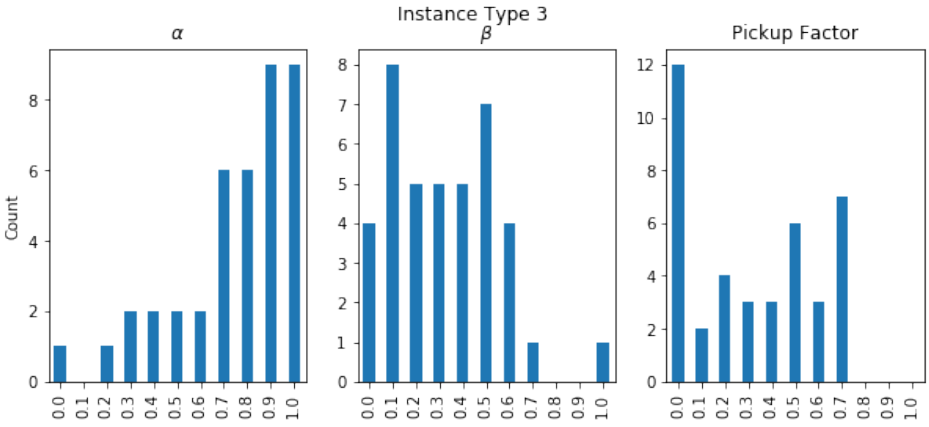

Figure 4: Occurrence of Weight of Instance Type 3

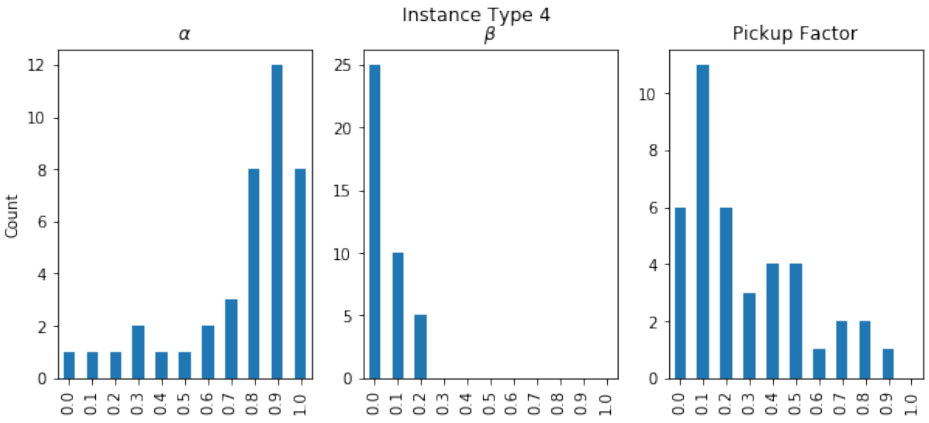

Figure 5: Occurrence of Weight of Instance Type 4

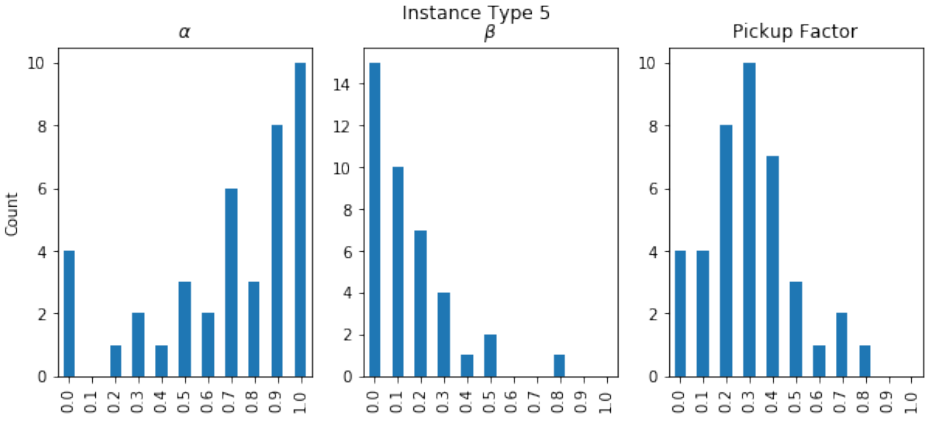

Figure 6: Occurrence of Weight of Instance Type 5

D Some Example Routes

In these figures, ● indicates a pivot and the p and d indicate pickups and deliveries respectively.

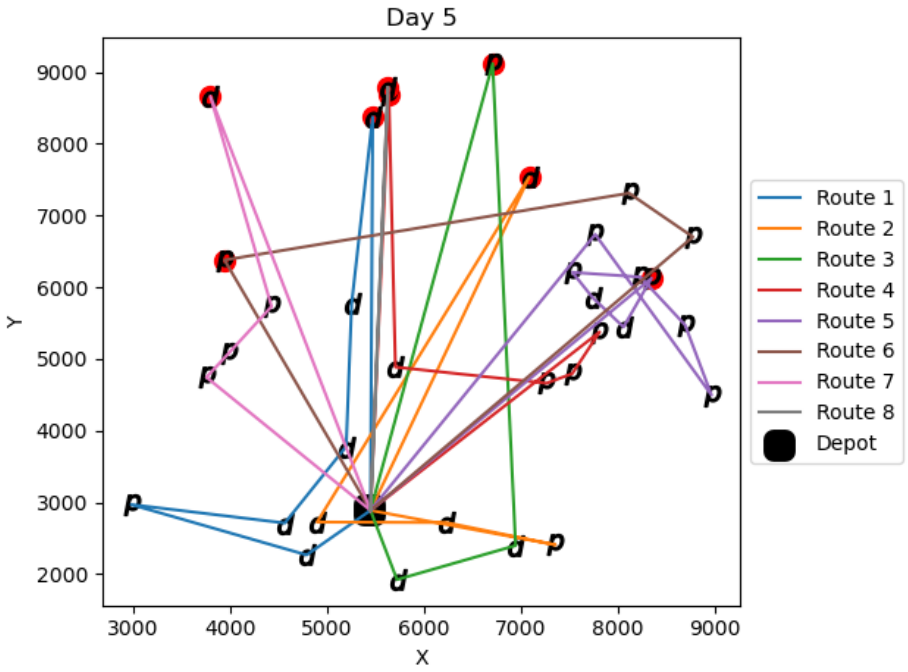

Figure 7: Route on day 5 of instance `r100d10_4`

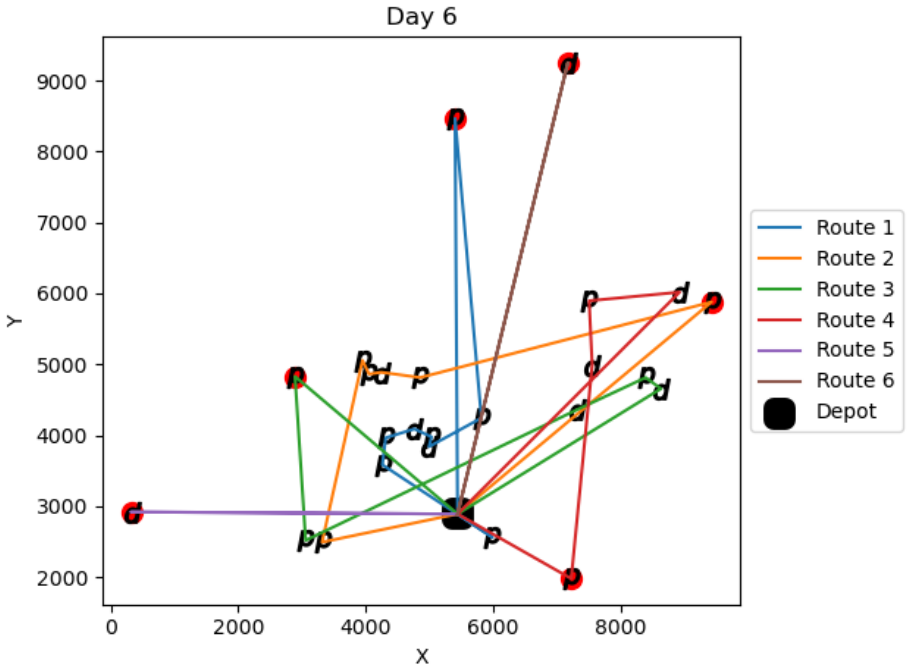

Figure 8: Route on day 6 of instance `r100d10_2`