Insertion Heuristics for a Class of Dynamic Vehicle Routing Problems

Matthew Randall[*] Ahmed Kheiri[†] Adam N. Letchford[†]

November 2022

Abstract

We consider a simple family of dynamic vehicle routing problems, in which we have a fixed fleet of identical vehicles, and customer requests arrive during the route-planning process. For this kind of problem, it is natural to use an insertion heuristic. We test several such heuristics computationally, on two different variants of the problem. It turns out that a parallel heuristic, based on a certain "sum-of-squares" insertion criterion, significantly outperforms the others.

Keywords: dynamic vehicle routing; insertion heuristics; parallel insertion

1 Introduction

Vehicle routing problems (VRPs) are a very well-known class of combinatorial optimisation problems, and there is a huge literature on them, including several books (e.g., [2, 11, 24]). An important distinction in the VRP literature is between *static* VRPs, in which all of the relevant data is known before the routes need to be planned, and *dynamic* VRPs, in which new information can come in during the route-planning process, or even after the vehicles have set off (e.g., [16, 18]). Dynamic VRPs tend to be much harder to solve than static ones, yet they have received less attention.

In this paper, we consider one specific dynamic situation, in which customer requests arrive one at a time, and the routes must be constructed as the requests come in. In this context, it is natural to use *insertion* heuristics. The idea of such a heuristic is that we start with a collection of "empty" routes, and then iteratively attempt to insert each new customer into one of the routes.

[*]STOR-i Centre for Doctoral Training, Lancaster University, Lancaster LA1 4YR, UK. E-mail: `M.Randall1@lancaster.ac.uk`

[†]Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK. E-mail: {`A.Kheiri,A.N.Letchford`}`@lancaster.ac.uk`

Insertion heuristics were first introduced for the TSP [20], and then extended to the VRP with time windows by Solomon [23]. Since then, many more insertion heuristics have been devised for various static VRPs (e.g., [22, 17, 21, 12, 14]).

Insertion heuristics do not give particularly good solutions for static VRPs, and they tend to be combined with local search heuristics or meta-heuristics [8, 10]. Nevertheless, insertion heuristics remain an attractive alternative for the dynamic VRPs that we are considering here. (In fact, in some circumstances, they may be the only option, because there may not be sufficient time between consecutive requests to perform local search.)

The aim of this paper is to gain more insight into the relative performance of various insertion heuristics in the dynamic setting. We consider five different insertion heuristics, which we call "sequential", "quasi-sequential", "naive parallel", "seeded parallel" and "sum-of-squares parallel". We also consider two different dynamic VRPs: a dynamic version of the capacitated VRP [7] and a dynamic version of the distance-constrained VRP [13]. It turns out that the sum-of-squares heuristic significantly outperforms the others for both problems.

The paper has the following structure. In Section 2, we define our two problems formally. In Section 3, we describe several different insertion heuristics. The computational results are given in Section 4. Finally, Section 5 contains some concluding remarks.

2 Two Dynamic VRPs

In this section, we explain our dynamic setting in more detail, and then define our two specific dynamic VRPs.

2.1 General setup

We have a single depot and a fixed fleet of m identical vehicles, all of which must start and finish their routes at the depot. There may also be side constraints, such as limited vehicle capacities or a restriction on the length of each route. The primary objective is to maximise the number of customers served, but the secondary objective is to minimise the total travel distance.

A total of n customers will place orders, but the value of n is not known in advance, nor are the locations of the customers. We assume that there is an "ordering period", during which customer orders are placed and routes are constructed simultaneously. We will view the ordering period as consisting of n "decision epochs", numbered from 1 to n. At the start of epoch e, the eth customer reveals their location, together with any other relevant information (such as demands or time windows). An insertion heuristic must then decide (a) whether it is possible to insert the order into one of the existing routes and, if so (b) where to insert the order.

To describe this in detail, we introduce some notation. For $e = 1, \ldots, n$, we let A_e denote the set of customers whose requests have been accepted by the end of epoch e. We also use the convention that $A_0 = \emptyset$. In epoch e, the insertion heuristic attempts to find a feasible way to insert customer e into one of current routes. If it is successful, the request of customer e is accepted, and A_e is set to $A_{e-1} \cup \{e\}$. Otherwise, the request is declined and A_e is set to A_{e-1}.

Now, for $i = 1, \ldots, m$ and $e = 1, \ldots, n$, we let $C_{i,e}$ denote the set of customers that have been allocated to the ith vehicle at the end of epoch e. We also use the convention that $C_{i,0} = \emptyset$ for $i = 1, \ldots, m$. Note that

$$A_e = \bigcup_{i=1}^{m} C_{i,e} \qquad (e = 0, \ldots, n).$$

In any given epoch, the current route for vehicle i is stored as an ordered sequence of nodes:

$$\mathbf{R}^{(i)} = \left(0, r_1^{(i)}, \ldots, r_{|C_{i,e}|}^{(i)}, 0\right)^T,$$

where $r_j^{(i)} \in \{1, \ldots, e\}$ represents the jth customer visited by vehicle i. Note that $r_0^{(i)} = r_{|C_{i,e}|+1}^{(i)} = 0$ denotes the depot, since each route starts and ends there. At the start of the heuristic, the routes are initialised as $\mathbf{R}^{(i)} = (0, 0)^T$. If the insertion heuristic accepts the customer's order during epoch e, the heuristic then selects a vehicle i, along with a position in that vehicle's route, and e is inserted into $\mathbf{R}^{(i)}$ in the given position.

With this notation, the primary objective is to maximise $|A_n|$, and the secondary objective is to minimise

$$\min \sum_{i=1}^{m} \sum_{j=0}^{|C_{i,n}|} d\left(r_j^{(i)}, r_{j+1}^{(i)}\right),$$

where $d(j, j') = d(j', j)$ denotes the distance between customers j and j', and $d(0, j)$ denotes the distance between the depot and customer j.

2.2 A dynamic capacitated VRP

Our first problem is a dynamic version of the well-known *capacitated* VRP [7]. We are given a positive integer Q, representing the vehicle capacity. Each customer will have a positive integer demand, which is not known in advance. The total demand of the customers served by any single vehicle must not exceed Q. In our notation, we require

$$\sum_{e \in C_{i,n}} q_e \leq Q \qquad (i = 1, \ldots, m),$$

where q_e denotes the demand of the eth customer.

We will call this problem the *dynamic capacitated VRP*, or DCVRP for short.

2.3 A dynamic distance-constrained VRP

In our second problem, we have a distance constraint instead of a capacity constraint (see [13]). More precisely, we are given a positive integer D, and the length of each vehicle route is not permitted to exceed D. In our notation, this amounts to imposing

$$\sum_{j=0}^{|C_{i,n}|} d\left(r_j^{(i)}, r_{j+1}^{(i)}\right) \leq D \qquad (i = 1, \ldots, m).$$

We will call this problem the *dynamic distance-constrained VRP*, or DDVRP for short.

3 Five Insertion Heuristics

The general framework mentioned in Subsection 2.1 permits one to define many different insertion heuristics, depending on the precise rule used to select the insertion point. In this section, we will consider five specific rules and discuss their pros and cons.

3.1 Sequential insertion

In *sequential* insertion, the vehicles are filled one at a time. We allocate as many customers as possible to vehicle $i = 1$, until we encounter a customer that cannot be inserted into the route of that vehicle (due to capacity or distance constraints). From that point, we allocate as many customers as possible to vehicle $i = 2$, and so on. The process continues until (a) there are no more vehicles available or (b) the ordering period has ended. (In the case of the DDVRP, we disregard any customers that cannot possibly appear on any route, i.e., any customer e for which $2d(0, e) > D$).

Now, suppose that we have found that customer e can be inserted into the ith route. If there are several possible insertion points, we simply choose the point which leads to the smallest increase in the length of the ith route. More precisely, we search sequentially through $\mathbf{R}^{(i)}$ to find the position j which minimises

$$d\left(r_j^{(i)}, e\right) + d\left(e, r_{j+1}^{(i)}\right) - d\left(r_j^{(i)}, r_{j+1}^{(i)}\right).$$

We remark that, although sequential insertion is a very simplistic heuristic, it may be necessary in real-world applications where orders can arrive even after some of the vehicles have been sent out.

3.2 Quasi-sequential insertion

Observe that it could happen that the current customer cannot be inserted into route i, but a later customer can be. (In the case of the DCVRP, this could happen if the current customer has a very large demand. In the case of the DDVRP, it could happen if the current customer is far away from route i.) This leads us to consider a modified version of sequential insertion, which we call *quasi-sequential*.

The idea is as follows. If it is possible to insert a customer into the first route, we do so. Otherwise, we check whether the customer can be inserted into the second route. If it is possible, we do so. And so on. The process continues until the end of the ordering period. If there are several possible insertion points in any given epoch, we again choose the point which leads to the smallest increase in the length of the given route.

3.3 Naive parallel insertion

Our third insertion rule is called *naive parallel* insertion. Here, we attempt to construct routes for all of the vehicles in parallel. More precisely, whenever a new customer requests a delivery, we check all possible insertion positions in all of the m routes. If a feasible insertion point exists, the request is accepted. If there are several feasible insertion points, we choose the point which leads to the smallest increase in the total length of the routes.

We call this rule *naive* for the following reason. In practice, we found that the naive insertion heuristic behaves in a more-or-less sequential manner. To see why, suppose that we have just inserted the first customer into the first route, and we are now considering where to insert the second customer. Due to the triangle inequality, it will almost always be cheaper to insert the second customer into the first route rather than into one of the other routes. This in turn will make it more likely that the third customer will be inserted into the first route as well. As a result, the routes tend to "fill up" one after another, and the resulting solution tends to be very similar to the one obtained with quasi-sequential insertion.

We will comment on this phenomenon again in Section 4.

3.4 Parallel insertion with seeds

Our fourth insertion rule is called *parallel insertion with seeds*. In this method, we "seed" the routes, in an attempt to prevent the behaviour mentioned in the previous subsection. More precisely, in the first m decision epochs, each customer is allocated to a different route. From that point on, we proceed as in naive parallel insertion.

3.5 Sum of squares insertion

Our fifth and final rule is inspired by the work of Bektaş and Letchford [3]. They observed that, in optimal solutions to (static) VRPs, some routes are often much longer than others. Such solutions will be perceived as unfair by the drivers. To address this, they proposed to minimise the sum of the *squared* route lengths rather than the sum of the route lengths.

Although [3] was concerned with static VRPs, one can apply the same concept to dynamic VRPs. To do this, we modify the naive insertion rule in the following way. If there are several feasible insertion points for a given customer e, we choose the point which leads to the smallest increase in the sum of the squared route lengths. That is, we check all vehicles i and positions j sequentially for that which minimises

$$\left(T_{i,e-1} + d\left(r_j^{(i)}, e\right) + d\left(e, r_{j+1}^{(i)}\right) - d\left(r_j^{(i)}, r_{j+1}^{(i)}\right)\right)^2 - T_{i,e-1}^2,$$

where

$$T_{i,e-1} = \sum_{k=0}^{|C_{i,e-1}|} d\left(r_k^{(i)}, r_{k+1}^{(i)}\right)$$

is the total travel distance of route i at the end of epoch $e-1$.

We call this last insertion rule *sum-of-squares* (SoS) insertion. We will see in the next section that the SoS insertion performs remarkably well.

4 Computational Results

In this section, we describe our computational experiments and present the results. Subsection 4.1 presents some preliminary remarks. Subsections 4.2 and 4.3 report results for the DCVRP with general and unit demands, respectively. Finally, Subsection 4.4 gives the results for the DDVRP.

4.1 Preliminary remarks

The five insertion heuristics were coded in C++ and compiled using g++. All experiments were run on an Intel Xeon® Gold 6248R processor running at 3.00 GHz with 128GB of RAM.

As far as we know, there are no benchmark instances available for the dynamic VRPs under consideration. Accordingly, we selected some *static* benchmark instances, and converted them into dynamic VRPs simply by assuming that the customer information is revealed one customer at a time.

It is important to note that the order in which the customers appear can affect the solution that is obtained by any of our insertion heuristics. For this reason, we apply each of our heuristics 1000 times on each instance, randomising the customer order each time.

Table 1: Average number of customers served for the DCVRP instances (to 2 dp). Values in bold mean that all customers were always served.

Instance	Q	Seq	QSeq	Naive	Seed	SoS
E-n13-k4	6000	**12.00**	**12.00**	**12.00**	**12.00**	**12.00**
E-n22-k4	6000	19.93	21.00	20.99	20.85	20.76
E-n23-k3	4500	21.24	**22.00**	**22.00**	21.70	21.56
E-n30-k3	4500	26.29	28.96	28.96	28.79	28.48
E-n31-k7	140	26.14	29.99	29.98	29.53	29.81
E-n33-k4	8000	31.15	**32.00**	32.00	31.98	32.00
E-n51-k5	160	48.48	50.00	50.00	49.89	49.69
E-n76-k7	220	**74.40**	**75.00**	**75.00**	**75.00**	**75.00**
E-n76-k8	180	74.40	**75.00**	**75.00**	75.00	74.99
E-n76-k10	140	71.31	74.99	74.93	74.45	74.20
E-n76-k14	100	**75.00**	**75.00**	**75.00**	**75.00**	**75.00**
E-n101-k8	200	100.00	**100.00**	**100.00**	**100.00**	**100.00**
E-n101-k14	112	98.24	**100.00**	**100.00**	100.00	100.00
F-n45-k4	2010	41.56	**44.00**	44.00	43.89	43.76
F-n72-k4	30000	**71.00**	**71.00**	**71.00**	**71.00**	**71.00**
F-n135-k7	2210	**134.00**	**134.00**	**134.00**	**134.00**	**134.00**
M-n101-k10	200	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
M-n121-k7	200	118.21	**120.00**	**120.00**	120.00	119.99
M-n151-k12	200	149.98	**150.00**	**150.00**	**150.00**	**150.00**
M-n200-k16	200	190.07	197.77	198.35	197.59	197.38
M-n200-k17	200	198.95	**199.00**	**199.00**	**199.00**	**199.00**

4.2 Results for the DCVRP with general demands

Our first set of DCVRP instances is based on the classical 'E', 'F' and 'M' CVRP instances, due to [5], [9] and [6], respectively. These instances are all available in the TSPLIB [19], and optimal solution values are known for all of them (see [15]). In these instances, customers have *general demands.* (This means that the demands may be of different sizes.)

Table 1 shows the average number of customers served for each instance and each heuristic, where the average is taken over 1000 random customer orders as mentioned above. The first column gives the instance name. (The convention is that "n" represents the total number of nodes, including the depot, and "k" represents the number of vehicles.) The second column shows the vehicle capacity. The columns headed "Seq" and "QSeq" correspond to the sequential and quasi-sequential heuristics, respectively. The remaining three columns correspond to the naive, seeded and SoS parallel heuristics. If a number is in bold, it indicates that the given heuristic always managed to serve all of the customers for the given instance, regardless of the order in which the customers were permuted.

It is clear that, for these instances, it is fairly easy to satisfy the majority

Table 2: Average total distance for the DCVRP instances. The best value for each instance is given in bold.

Instance	Stat	Seq	QSeq	Naive	Seed	SoS
E-n13-k4	247	354	346	343	319	**313**
E-n22-k4	375	613	626	612	490	**441**
E-n23-k3	569	786	771	748	631	**617**
E-n30-k3	534	775	792	772	624	**554**
E-n31-k7	379	862	904	756	672	**666**
E-n33-k4	835	1100	1104	1088	**1017**	1076
E-n51-k5	521	964	979	966	746	**646**
E-n76-k7	682	1446	1422	1399	980	**836**
E-n76-k8	735	1553	1554	1519	1077	**933**
E-n76-k10	830	1698	1742	1687	1191	**1064**
E-n76-k14	1021	1814	1743	1691	1135	**1087**
E-n101-k8	815	1800	1777	1736	1185	**1024**
E-n101-k14	1067	2381	2359	2251	1503	**1396**
F-n45-k4	724	1333	1315	1227	979	**860**
F-n72-k4	237	545	484	457	428	**293**
F-n135-k7	1162	2844	2673	2596	1809	**1648**
M-n101-k10	820	2346	2290	2275	1399	**1180**
M-n121-k7	1034	2168	2177	2167	1901	**1792**
M-n151-k12	1015	2645	2608	2547	1613	**1411**
M-n200-k16	1274	3376	3350	3354	2086	**1884**
M-n200-k17	1275	3559	3460	3384	2043	**1866**

of the customers. One can check that the mean percentages of customers served by the five heuristics, taken over all of the instances, were 97.36%, 99.98%, 99.97%, 99.70% and 99.59%, respectively. Thus, all heuristics apart from the sequential one perform extremely well. Note also that the quasi-sequential and naive parallel heuristics found solutions of very similar quality. This is due to the phenomenon mentioned in Subsection 3.3.

Table 2 reports the average total distance of the solutions found by the heuristics. Here, there is an extra column, labelled "Stat", which gives the total distance for the optimal solution to the static version of the problem. In the last five columns, each figure is rounded to the nearest integer. The best value for each instance is given in bold.

As one would expect, the heuristic solutions to the dynamic CVRP are significantly worse than the optimal solution to the static version. Among the five heuristics, the SoS version is the winner, with the seeded version a close second. In fact, the solutions found by the seeded version were on average 14.7% longer than the ones found by the SoS version, and the solutions found by the other three heuristics were over 50% longer on average.

It might be objected that the distance figures are not comparable, given that the heuristics sometimes failed to serve every customer. So, for interest, we computed the *mean distance travelled per customer served* for each heuristic and each instance. The average figures for the five heuristics, over all instances, were 23.9, 23.0, 22.2, 17.1 and 15.8, respectively. So, also using this measure, the seeded and SoS versions come out top. (For interest, the corresponding figure for the optimal static solution was 12.3.)

In order to gain more insight into the behaviour of the five insertion rules, we computed some more statistics for the largest instance, M-n200-k17. For each of the five rules, and for each epoch of the ordering period, we computed the *mean number of vehicles in use* (i.e., the mean number of vehicles that had at least one customer assigned to them). Figure 1 shows the results, where each curve corresponds to a different insertion rule. The curve at the bottom, which is blue in the online version of this paper, corresponds to the sequential insertion rule. As expected, it fills up the routes one at a time. The next curve up, which is pink in the online version, represents the quasi-sequential and naive parallel insertion rules. (The two methods give almost identical results for this instance.) This curve is only slightly higher than the bottom one, showing that the quasi-sequential and naive parallel insertion rules behave in an "almost sequential" manner. The next curve up, which is black in the online version, represents the SoS insertion rule. It is apparent that more vehicles are in use throughout the ordering period. Finally, the curve at the top, which is green in the online version, corresponds to the seeded parallel insertion rule. It inserts the first m customers into different vehicles, as it is designed to.

4.3 Results for the DCVRP with unit demands

Next, we applied the heuristics to DCVRP instances with *unit demands.* (That is, every customer has a demand equal to 1.) These instances were based on the static unit-demand CVRP instances described in Araque *et al.* [1]. (They were kindly provided to us by Jens Lysgaard.)

Note that, in the unit-demand case, all five heuristics will always serve exactly $\min\{n, mQ\}$ customers. Thus, for these instances, we report only the total travel distance. Moreover, the sequential and quasi-sequential heuristics will always generate the same solution, since the first Q customers will be inserted into the first route, the next Q into the second route, and so on.

Table 3 shows some statistics for these instances, along with the average total distances. The first five columns show the instance name, number of customers, number of vehicles, capacity Q, and the optimal total distance for the static problem. The column headed "Seq" contains the results for the sequential and quasi-sequential heuristics. The other columns are as in Table 2. As before, the best value for each instance is given in bold.

Here, there is a clear hierarchy, with the SoS rule giving the best results by far, and the seeded rule coming second. Moreover, as before, the

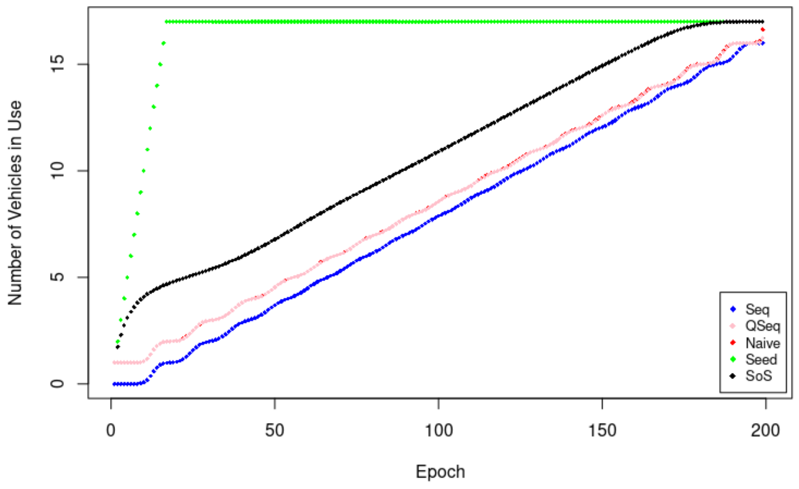

Figure 1: Mean number of vehicles in use at each epoch with various insertion methods for instance "M-n200-k17".

Table 3: Average total distance for the unit-demand DCVRP instances. The best value for each instance is given in bold.

Instance	n	m	Q	Stat	Seq	Naive	Seed	SoS
AKMP1	40	4	10	647	1183	1183	920	**818**
AKMP7	50	7	8	875	1777	1777	1210	**1076**
AKMP8	50	4	15	678	1296	1296	931	**780**
AKMP13	60	2	30	688	1050	1050	923	**850**
AKMP15	21	7	3	530	830	830	668	**631**
AKMP16	21	3	7	341	540	540	445	**414**
AKMP17	29	8	4	832	1524	1524	1080	**1037**
AKMP18	29	5	6	639	1176	1176	885	**819**
AKMP19	32	7	5	627	1059	1059	821	**741**
AKMP20	32	4	8	497	811	811	682	**610**

Table 4: Average number of customers served for the DDVRP instances (to 2 dp). The best value for each instance is given in bold.

Instance	n	Seq	QSeq	Naive	Seed	SoS
AKMP1	40	10.72	30.92	33.40	34.61	**37.09**
AKMP7	50	10.95	36.62	38.40	37.56	**39.90**
AKMP8	50	10.22	33.2	35.76	37.64	**40.95**
AKMP13	60	21.81	36.43	37.48	45.06	**49.80**
AKMP15	21	10.68	16.26	16.39	15.17	**16.41**
AKMP16	21	7.03	16.66	17.31	17.36	**18.22**
AKMP17	29	14.44	18.00	18.00	**19.29**	18.00
AKMP18	29	8.59	21.53	21.91	20.82	**22.14**
AKMP19	32	12.38	27.00	27.81	26.51	**28.35**
AKMP20	32	11.45	27.22	28.38	28.44	**29.91**

(quasi-)sequential and naive parallel heuristics found solutions of very similar quality.

4.4 Results for the DDVRP

Finally, to construct our DDVRP instances, we simply took the ten Araque *et al.* instances and replaced the capacity constraint with a distance constraint. To ensure that the distance constraint was a genuine restriction, we set D to $\lceil OPT/m \rceil$, where OPT is the figure displayed in the fifth column of Table 3. Note that the optimal solutions of these new instances are not known, not even for the static version of the DVRP.

Table 4 shows the average number of customers served for these instances. The format is similar to that of the previous tables. The best value for each instance is given in bold. Once again, the "SoS" heuristic is the clear winner.

Finally, as in the previous subsection, we computed the *mean distance travelled per customer* for each heuristic and each instance. The average figures for the five heuristics were 45.4, 24.1, 23.1, 23.4 and 21.4, respectively. Here, again, the SoS variant performs very well. We remark that the extremely poor performance of the sequential variant here is caused by the presence of a few customers that cannot be served at all, because they are too far from the depot.

It remains to be explained why the SoS heuristic performs so well. A possible explanation follows. By penalising long routes, the SoS rule causes the routes to have similar lengths throughout the ordering period. As a result, none of the routes are likely to be completely full until near the end of the ordering period. Thus, for all customers except the last, there tend to be many feasible insertion positions. This increased number of options allows for more efficient insertion of the later customers, which tends to outweigh the slightly less efficient insertion of the earliest customers.

5 Conclusions

We tested five different insertion heuristics for dynamic VRPs. The main conclusions are (a) if a parallel insertion heuristic is implemented in a naive way, it is likely to perform as poorly as the sequential version, and (b) good results are obtained by inserting customers in the position that minimises the sum of the *squared* route lengths.

An interesting topic for future research is whether the "sum-of-squares" insertion heuristic can be adapted to more complex dynamic VRPs, such as ones in which each prospective customer must be offered a selection of possible time windows (see [4]).

Acknowledgements: The authors thank Jens Lysgaard for sharing the AKMP instances with us. The first author gratefully acknowledges financial support from the EPSRC (through the STOR-i Centre for Doctoral Training under grant EP/S022252/1), and from Tesco PLC.

References

[1] J.R. Araque, G. Kudva, T.L. Morin, and J.F. Pekny. A branch-and-cut algorithm for vehicle routing problems. *Ann. Oper. Res.*, 50:37–59, 1994.

[2] M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors. *Network Routing.* North Holland, Amsterdam, 1995.

[3] T. Bektaş and A.N. Letchford. Using ℓ^p-norms for fairness in combinatorial optimisation. *Comput. Oper. Res.*, 120: article 104975, 2020.

[4] A.M. Campbell and M.W.P. Savelsbergh. Incentive schemes for attended home delivery services. *Transp. Sci.*, 40:327–341, 2006.

[5] N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *J. Oper. Res. Soc.*, 20:309–318, 1969.

[6] N. Christofides, A. Mongozi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

[7] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Manag. Sci.*, 6:80–91, 1959.

[8] R. Elshaer and H. Awad. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Comput. Ind. Eng.*, 140: article 106242, 2020.

[9] M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Oper. Res.*, 42:626–642, 1994.

[10] M. Gendreau, J.-Y. Potvin, O. Bräysy, G. Hasle, and A. Løkketangen. Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography. In B.L. Golden, S. Raghavan, and E.A. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 143–169. Springer, Boston, MA, 2008.

[11] B.L. Golden, S. Raghavan, and E.A. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges.* Springer, Boston, MA, 2008.

[12] G. Ioannou, M. Kritikos, and G. Prastacos. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *J. Oper. Res. Soc.*, 52:523–537, 2001.

[13] G. Laporte, M. Desrochers, and Y. Nobert. Two exact algorithms for the distance constrained vehicle routing problem. *Networks*, 14:161–172, 1986.

[14] K.-W. Pang. An adaptive parallel route construction heuristic for the vehicle routing problem with time windows constraints. *Expert Syst. Appl.*, 38:11939–11946, 2011.

[15] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Math. Program. Comput.*, 9:61–100, 2017.

[16] V. Pillac, M. Gendreau, C. Guéret, and A.L. Medaglia. A review of dynamic vehicle routing problems. *Eur. J. Oper. Res.*, 225:1–11, 2013.

[17] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Eur. J. Oper. Res.*, 66:331–340, 1993.

[18] H.N. Psaraftis, M. Wen, and C.A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67:3–31, 2016.

[19] G. Reinelt. TSPLIB–a traveling salesman problem library. *ORSA J. Comput.*, 3:376–384, 1991.

[20] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 1977.

[21] R.A. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transp. Sci.*, 29:156–166, 1995.

[22] M.W.P. Savelsbergh. A parallel insertion heuristic for vehicle routing with side constraints. *Stat. Neerl.*, 44:139–148, 1990.

[23] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35:254–265, 1987.

[24] P. Toth and D. Vigo, editors. *Vehicle Routing: Problems, Methods, and Applications.* SIAM, Philadelphia, PA, 2014.