CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Efficiently Solving Very Large Scale Routing Problems

Florian Arnold
Michel Gendreau
Kenneth Sörensen

December 2017

CIRRELT-2017-75

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Efficiently Solving Very Large Scale Routing Problems

Florian Arnold[1,*], Michel Gendreau[2], Kenneth Sörensen[1]

[1] University of Antwerp, Department of Engineering Management, ANT/OR, Operations Research Group, Stadscampus B-building, Prinsstraat 13, 2000 Antwerpen, Belgium

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

Abstract. The vehicle routing problem is one of the most-studied and practically relevant problems in combinatorial optimization. However, even though some applications like waste collection and parcel distributions can require thousands or even tens of thousands of customer visits, almost all research thus far has been targeted at solving problems of not more than a few hundred customers. In this paper we develop a heuristic for the capacitated vehicle routing problem that is able to compute high-quality solutions for such very large scale instances in a reasonable computing time. We demonstrate how to linearize the time and space complexity of a heuristic that is entirely based on a well-implemented local search component. The resulting algorithm outperforms a previously introduced heuristic by a significant margin, and computes high-quality solutions for instances of 10,000 and more customers in less than an hour. In order to stimulate more research in this domain, we introduce a new set of benchmark instances that are based on a real-world problem.

Keywords. Vehicle routing problems, heuristics, metaheuristics, very large scale.

---

* Corresponding author: florian.arnold@uantwerpen.be

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2017

1. Introduction

Routing problems form one of the most studied areas in combinatorial optimization. Among these, the Travelling Salesman problem (TSP) and the Vehicle Routing Problem (VRP) and its variants have resulted in thousands of publications and a plethora of innovative optimization techniques over the last decades. Their popularity stems both from their complexity (categorized as NP-hard) and from their applicability in practice, both of which have stimulated a large number of contributions. Routing problems share a common problem formulation: starting from a source (also called depot) a number of destinations (customers) should be visited in such a way that the length of the resulting tour(s) is minimized. In the canonical vehicle routing problem (also called the capacitated VRP or CVRP), the capacity of the used vehicles is limited, and, thus, more than one tour is usually necessary to visit all customers. A more formal introduction can be found in (Laporte, 2007). Due to the complexity of this combinatorial optimization problem, only small instances of up to 200 customers can be solved to optimality by exact methods. Therefore, a major research stream on the design of heuristics for the CVRP has evolved, the aim of which is to find high-quality solutions in a reasonable amount of computing time.

Recent research has focused on developing algorithms that are able to solve routing problems of at most a few hundred customers. The reason for this focus on relatively small instances is unclear. One potential explanation is that it is a result of the relatively small size of most popular benchmark instances. Indeed, the instances by Golden et al. (1998) contain up to 483 customers, and the largest problems in the more recent benchmark set by Uchoa et al. (2017) has 1001 customers. Since a good performance on the standard benchmark instance sets significantly increases a paper's publication potential, it is natural to see heuristics developed that specifically target this problem size.

Even though routing problems of up to 1000 customers cover a wide range of real-world applications, there are routing problems of significantly larger sizes that need to be tackled on a daily basis. One example presented in Kytöjoki et al. (2007) is waste collection, where trucks have to collect waste from tens of thousands of customers. With more and more information, about, e.g., the amount of waste in each container becoming available and with increasing standards in service level and cost pressure, there is a need to model and optimize these large-scale systems more accurately. Another example is the distribution of parcels. The continuing growth of e-commerce goes hand in hand with a steady increase in the number of parcels that need to be delivered to customers every day. In Belgium, for example, the daily quantity of delivered parcels correspond to about 1% of the population, so that in cities like Brussels 20,000 and more deliveries need to be carried out every day. Given that the delivery business is largely cost-driven, these deliveries have to be planned and executed as efficiently as possible.

Consequently, there is a strong practical motivation to develop solution methods that can tackle much larger vehicle routing problems than those available in the standard benchmark instance sets. In the following we will use the term '*very large scale*'

to denote such instances since 'large scale' is commonly used to describe problems of several hundred customers. A first, and so far only, step in this direction has been taken a decade ago in Kytöjoki et al. (2007), who develop a variable neighborhood search algorithm that is able to solve instances of up to 20,000 customers. This limited amount of research is especially striking given that the literature on the TSP has covered very large instances since several decades (Reinelt, 1991), solving instances with up to 25,000,000 nodes. Of course, the development and testing of heuristics that are able to tackle very large instances requires that a benchmark set with such instances is available.

In this paper, we develop a highly-efficient, well-scaling, and yet simple heuristic that is able to compute high-quality solutions for problems of several thousand and more customers in a reasonable amount of computing time. The heuristic is entirely based on three well-implemented local search operators that have an almost linear computational complexity; it also outperforms the only other heuristic developed to solve very large instances. As a second contribution we generate 10 very large scale VRP instances. These instances are based on parcel demand data in Belgium and, thus, reflect real-world problems. The instance set is publicly available to stimulate more research in the domain of very large scale VRPs.

2. The challenges of very large scale VRPs

In recent years, solution methods have been primarily developed with the purpose of tackling VRPs of not more than a few hundred customers. In theory, the same methods could also be applied to solve problems of larger size, given the fact that they are mostly heuristics and should therefore scale reasonable well. In practice however, problems that are much larger, i.e., several thousands or tens of thousands of customers cannot be effectively solved by existing methods, as the required computation times of these methods would exceed any reasonable time limit. We demonstrate this effect by plotting the computation times of two state-of-the-art heuristics, the hybrid genetic heuristic (HGSDAC) by Vidal et al. (2013) and the iterated local search (ILS) by Subramanian et al. (2013) against problem size in Fig. 1, as reported in Uchoa et al. (2017). If we were to extrapolate these trajectories, we would conclude that the solution of a problem of, e.g., 10,000 customers (while keeping a similar solution quality) would require several months of computation time.

A similar observation can be made for the usage of memory. For instance, most heuristics store the complete distance matrix, i.e., the distance between each pair of nodes, in order to compute the length of edges and routes during the execution of the algorithm. This results in n^2 entries (where n is the number of nodes), which may exceed the available computer memory for large instances. If a distance entry is represented by 4 bytes, a distance matrix for a 20,000 customer instance would require almost 3GB memory. In addition to the distance matrix, there might be other data structures and pre-processed information that further increase memory usage way beyond available memory.

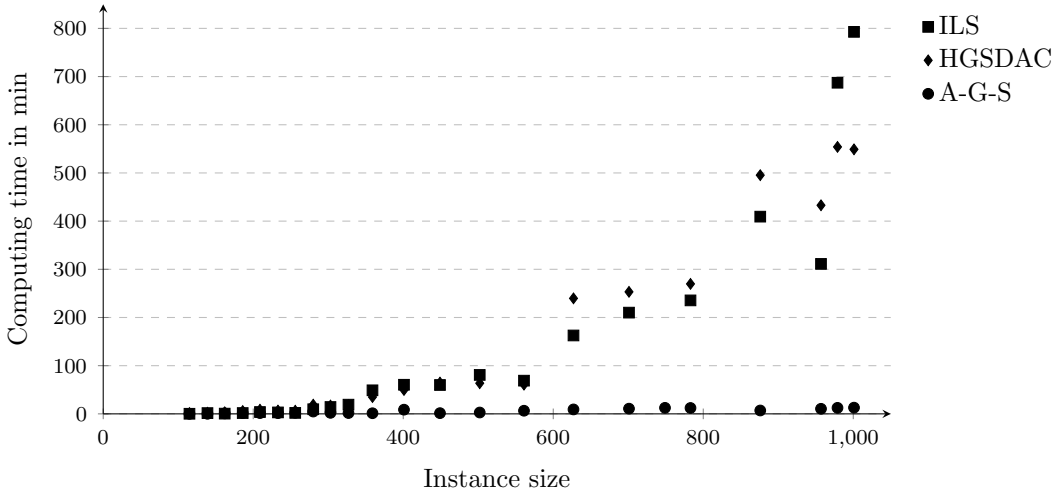

Figure 1: *Computation time as a function of of instance size on the instances of Uchoa et al. (2017) for two state-of-the-art heuristics from the literature (ILS and HGSDAC) and the heuristic presented in this paper (A-G-S). The data shows that the runtime of both heuristics grows in a nonlinear fashion, and much faster than the runtime of the presented heuristic. All three heuristics produce high-quality solutions on these instances (with average gaps for HGSDAC 0.24%, ILS 0.72%, A-G-S 0.61%).*

The solution to address these challenges, as we will demonstrate in this paper, is to decrease the runtime complexity and the memory usage in such a way that the algorithm runs in approximately linear time and linear space. If the runtime grows about linearly in the problem size, one can solve arbitrarily large instances in a reasonable time. In Fig. 1 we observe that the state-of-the-art heuristic exhibit a nonlinear runtime behavior and are, therefore, not suitable for very large scale instances. The same applies for the memory usage. In the above example of the distance matrix, the memory usage grows as a quadratic function of the problem size and quickly spirals out of control.

A widely-used approach to reduce (and possibly linearize) computational complexity is *heuristic pruning*, especially in the context of local search. Local search tries to iteratively improve a solution using small (local) modifications, called moves. By reducing the number of evaluated moves, the required computational effort to determine the next move can be considerably reduced. As an example, consider the popular type of move called *relocate*, which tries to remove a customer from its current route and re-insert it into the same or another route in a different position. If we try to relocate each customer into every possible position of every other route, we obtain a computational complexity of $O(n^2)$. This complexity can be reduced by only considering promising options for a re-insertion. For instance, if we only consider the 10 nearest nodes, we obtain a complexity of $10n \in O(n)$. Similar pruning techniques are used in many heuristics for the TSP (most notably by Lin and Kernighan (1973)) and the VRP (Mester and Bräysy, 2007). A metaheuristic that makes heavy use of pruning is granular tabu search (Toth and Vigo, 2003). In the heuristic described below, we will

use this technique in all of our local search operators.

A pruning technique that is especially worth mentioning is the one used by Helsgaun (2000) in the context of the TSP. The author argues that only considering a fixed number of nearest neighbours for each node is a rather coarse sieve. It might filter edges that are necessary for high-quality solutions and keep edges that are not worthwhile investigating. He suggests a finer measure of nearness that is based on the concept of a 1-tree, a spanning tree with one additional edge. The decision of whether to consider a certain edge in a TSP solution is based on its nearness, defined as the cost difference between the optimal 1-tree of all nodes and the minimum 1-tree that contains this edge. Since every tour is a 1-tree, both problems have a certain degree of similarity, but a minimal 1-tree can be computed more efficiently than a TSP solution. Thus, the cost differences in 1-trees give a good intuition about the likelihood that an edge will be included in an optimal tour. This idea has resulted in one of the most efficient heuristics for the TSP to date, which can solve problems with 100,000 or more customers.

Similar approaches can be used to linearize memory usage. Rather than storing the distance between each pair of customers, it might be sufficient to only store those distance values that are actually requested within the heuristic. For instance, it is extremely unlikely that the value of the distance between a node and its farthest-away neighbour will ever be retrieved, because this edge will simply not be taken into consideration in any promising solution (especially if there are 9,999 better alternatives). Thus, it might be sufficient to store only the distance values between relatively close nodes. If we only store information about the 100 closest neighbours, we reduce the memory usage from n^2 values to $100n$ values which results in a 100-fold memory reduction for 10,000 nodes, and — importantly — results in a linear rather than a quadratic memory usage as a function of the problem size. This idea is used by (Helsgaun, 2000), where only the distances between candidate edges (computed with the approach above) are stored, and all other requested distance values are computed and cached during runtime. A similar technique is used in (Applegate et al., 2003) where a neighbour graph is used for memory-representation of large scale TSPs. Kytöjoki et al. (2007) choose another approach. Rather than limiting the amount of stored information, the information is stored in a highly compressed format. Distance values are replaced by a more condensed representation of speed and twisty-road-curve factors, and the exact distances are recomputed during runtime. Of course, an optimal memory usage should make use of both approaches, limiting the amount of stored information to that which is actually needed, and representing this information in a format that is as compact as possible.

3. A Heuristic for very large scale VRPs

3.1. The knowledge-driven VRP heuristic of Arnold and Sörensen (2017)

The heuristic described in this paper is based on a heuristic to solve standard-size instances of the VRP. In the following, we present a brief outline of this VRP heuristic

which is described in more detail in (Arnold and Sörensen, 2017). We then explain how we adapt this heuristic to tackle very large scale instances.

The heuristic uses the Clarke and Wright heuristic (Clarke and Wright, 1964) to generate an initial solution, which is then iteratively improved. The improvement phase forms the core of the heuristic and is based on three existing complementary local search operators, which are implemented as efficiently as possible, and which perform both intra- and inter-route optimization. Individual routes are optimized with the Lin-Kernighan heuristic (Lin and Kernighan, 1973). For optimization between two routes we use the CROSS-exchange operator (CE) (Taillard et al., 1997). Optimization of two or more routes simultaneously is achieved using an ejection chain (EC) operator (Glover, 1996). The functioning of the CE and EC-operators is depicted in Figs. 2 and 3.

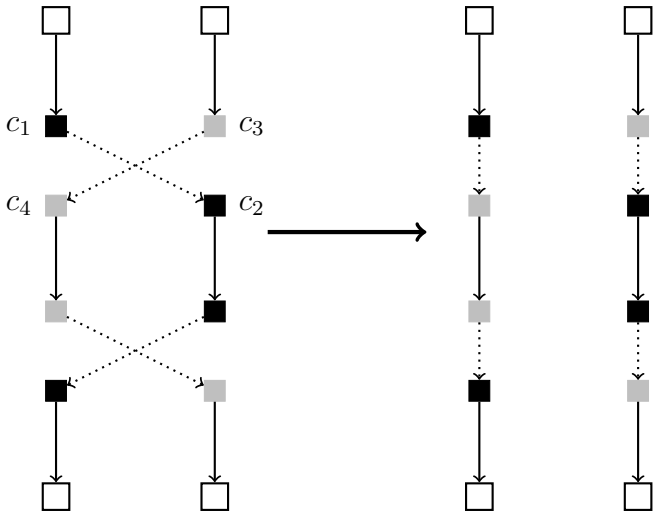

Figure 2: *Illustration of a CROSS-exchange move, where sequences of two customers are exchanged between two routes. The operator can be implemented to run in almost linear time, if we know which edge we have to remove (c_1, c_2) as a starting point of the operator. As potential new edges we only consider the nearest neighbours of c_1 that are currently in a different route, e.g., c_4. Edge (c_3, c_2) follows automatically, and it only remains to find two sequences starting from c_4 and c_2 that result in an overall saving when exchanged.*

The main idea behind this choice is that the EC and the CE operators generate large neighborhoods and are, therefore, usually successful in finding improving moves. On the downside, the generation of these neighborhoods usually comes at the cost of a high computational complexity that renders both operators impractical for larger problem sizes. However, we implement these two operators in such a way that they exhibit an almost linear runtime behavior. This complexity reduction is achieved by focussing on particular edges in the local search, rather than on the entire solution. More specifically, we identify edges that appear to be 'bad' and try to remove those by using them as the starting point for EC and CE. Consequently, rather than trying to find improvements everywhere, we target the local search at very specific areas of the

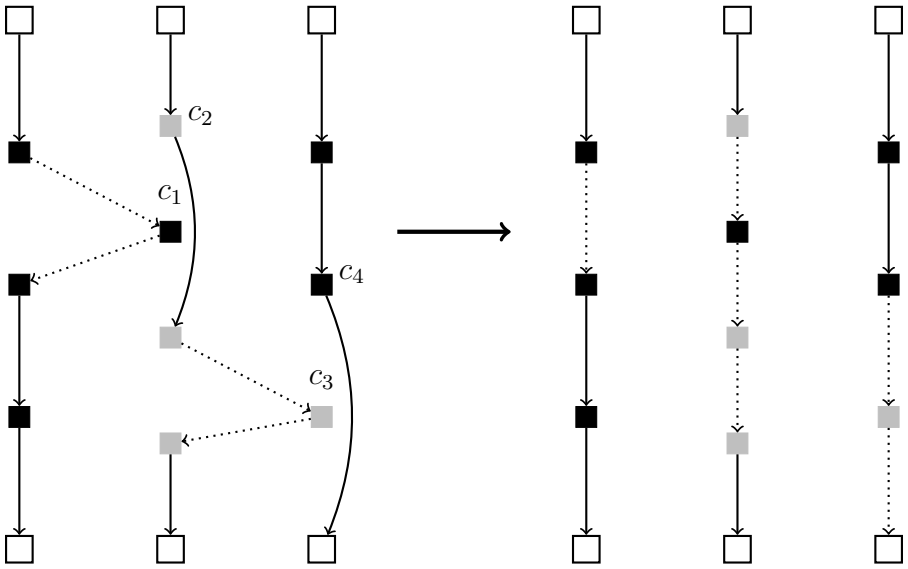

Figure 3: *Illustration of an ejection chain with two relocations. The operator runs in almost linear time if we know with which node to start the chain (c_1), and only consider close-by nodes as insertion points (c_1 next to c_2 and c_3 next to c_4). The costs associated with these insertions can be preprocessed to further enhance efficiency.*

solution space.

Those 'bad edges' are identified on the basis of problem-specific knowledge that we generated in a previous data-mining study (Arnold and Sörensen, 2016). It turned out that, rather than only removing long edges (or edges with a high *cost*) as suggested in Mester and Bräysy (2007), the removal of so-called *wide* edges is even more promising to obtain better solutions in a shorter time. Also, the *depth* of an edge can also be a good indicator of its badness. The metrics cost, width, and depth are illustrated in Fig. 4. In each iteration, the edges identified in this fashion are penalized by increasing their cost value, and used subsequently as the starting point for EC and CE. Whenever changes between routes are made, we use the Lin-Kernighan heuristic to re-optimize these routes separately. After a certain number of these iterations we apply the local search to all edges at the same time to optimize the solution globally. We refer to this final step as *global optimization*. This concept of edge penalization and local search is based upon the guided local search (GLS) metaheuristic (Voudouris and Tsang, 1999). For more details on the heuristic, we refer to Arnold and Sörensen (2017).

The resulting heuristic is simple in design, and yet is able to compete with the best heuristics in the literature on a wide range of benchmark instances. In a few minutes, it computes high-quality solutions with a gap of 0.70% and less for instances of up to 1,000 customers, which makes it up to 40 times faster than other state-of-the-art heuristics. In the original heuristic, the complexity of both the EC and CE operators is reduced by heuristic pruning, and only the nearest neighbors (usually only the 30 nearest neighbours) are considered in local search moves. Furthermore, much of the

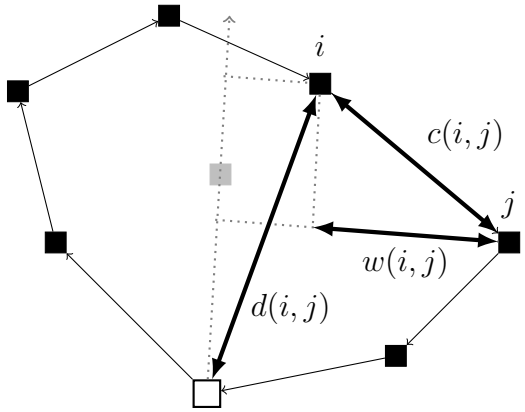

Figure 4: *Metrics determining the 'badness' of edge* (i, j)*. The* width $w(i, j)$ *is computed as the distance between nodes* i *and* j *measured along the axis perpendicular to the line connecting the depot and the route's center of gravity (gray dot). The* depth $d(i, j)$ *of this edge is the distance of the furthest node to the depot. The* cost $c(i, j)$ *is equal to the length of the edge.*

required information for the EC operator is preprocessed and stored to further minimize computational effort. As a result, we obtain a heuristic that scales more or less linearly in the instance size. This linear scaling behavior is a suitable basis to extend the heuristic to solve instances of larger magnitudes.

3.2. Adapting the heuristic of Arnold and Sörensen (2017) for very large scale instances

For the solution of instances of very large scale instances, i.e., instances of several thousand customers, several components of the original heuristic of Arnold and Sörensen (2017) need to be adapted. For each component, we therefore investigate both its computational complexity and memory usage. We find that intelligent heuristic pruning of the local search components and limiting the amount of data that is pre-processed and stored is sufficient to allow the heuristic to tackle very large scale instances. In this section, we describe the general principles behind these pruning and information storage strategies. In Section 4, we perform a series of detailed computational experiments.

The EC operator scales in a worse way than the other two operators and, depending on the specific instance, it can exhibit a nonlinear behavior even when configured as mentioned above. The reason for this nonlinear behavior lies in the nature of the operator. Starting with a relocation of one node into another route, it tries to add another relocation from this route and so forth, in other words it builds a chain of relocations. For each relocation after the initial one, we have several options for continuing relocations. We can try to relocate any node in the destination route next to each of its nearest neighbors in another route. Thus, with a growing number of subsequent relocations, the number of potential options grows exponentially. A straightforward solution to this problem is to limit the number of subsequent relocations that a single EC operation can perform. We observe that ejection chains of size three already exceed any reasonable computation time for larger instances and we, thus, limit the EC

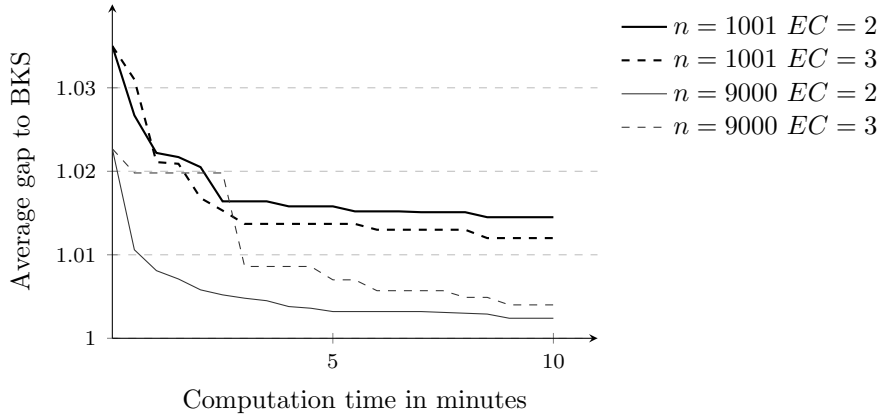

Figure 5: *Average gap (in %) to the best known solutions over time on two different instances with 1001 and 9000 customers. For large scale instances a lower length of the Ejection Chain (EC) has a better performance.*

operator to consider only two subsequent relocations. This effect is demonstrated on two instances in Fig. 5.

The CE operator already has a sufficiently low complexity, if implemented as in the heuristic of Arnold and Sörensen (2017). However, its runtime might still grow exponentially if the number of customers per route becomes very large. Since the CE operator tries to exchange two sequences in two different routes, its runtime is determined by the number of potential sequences that are investigated. In some of the very large scale instances that we considered, routes visit 500 and more customers, and thus, in the worst case the operator needs to investigate 500^2 sequence exchanges. Again, the solution to this problem is to impose a limit on the length of considered sequences. In our case we found that an upper bound of 100 nodes per sequence is sufficient to have a more or less linear runtime behavior with negligible loss in performance.

The Lin-Kernighan heuristic has an excellent scaling performance, and is used as the basis to tackle large scale TSPs of tens of thousands of customers (see, for instance, (Applegate et al., 2003) and (Helsgaun, 2000)). Therefore, not many changes are necessary to adapt it to very large scale VRP instances. As with the CE operator, we only encounter problems if routes become long and contain several hundred customers. In this case, we only consider the 10 nearest neighbors when performing *k-opt* moves (the original heuristic considered only the 5 nearest neighbors (Lin and Kernighan, 1973)). Additionally, we stop the heuristic after 4-opt (so we do not consider moves in which more than 4 edges are changed).

Finally, the heuristic constructs an initial solution with the Clarke-Wright heuristic (Clarke and Wright, 1964). This heuristic builds routes on the basis of a savings list that contains, for each pair of nodes, the savings that could be obtained when connecting these nodes. Thus, the savings list grows quadratically in the instance size, which makes this standard version inefficient for larger problems. We circumvent this problem by resorting to heuristic pruning once more. Relying on the assumption that good solutions

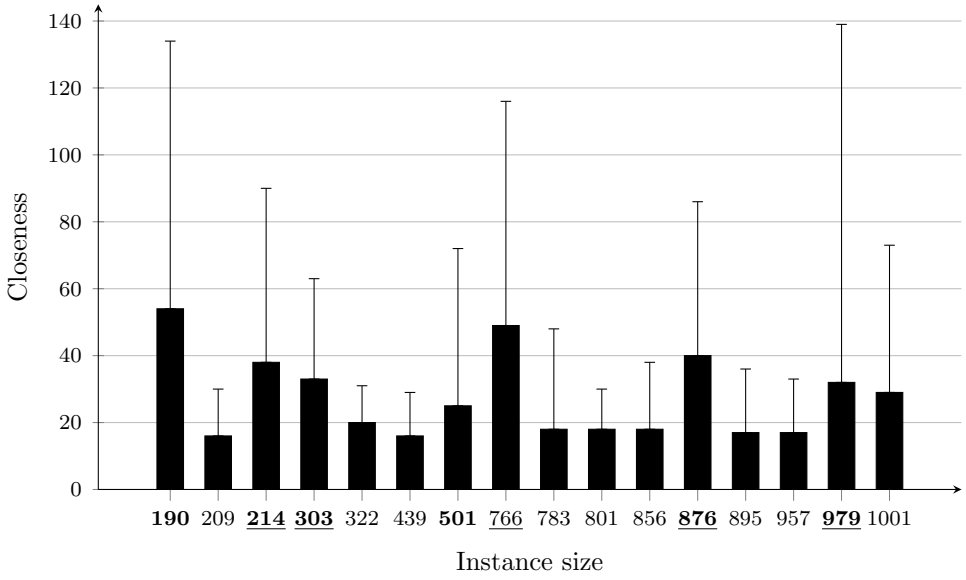

Figure 6: *Maximal closeness of 95% of the customers (bars), as well as maximal closeness of 99% of the customers (error bars) for different instances. Instances with a high variety in demand (1-100) are underlined, instances with clusters are highlighted in bold.*

typically only contain edges between close neighbors, we only compute the savings between a node and its 100 nearest neighbors, thereby linearizing the length of the list and the number of computational steps.

With the above adaptations, we have ensured that the runtime of the heuristic grows approximately linearly in the instance size, even when facing instances with extremely long routes. As the last step, we also need to linearize the memory usage. As suggested above, this can be achieved by limiting the amount of information stored to that which is actually needed during runtime. Typically, distance entries between far-away nodes are never retrieved because the corresponding edges will not be taken into consideration, so there is no need to store their value. In other words, we can drastically reduce the amount of used memory by asking the question: "Which edges will actually be considered during runtime?".

We approached this question by investigating the relative "closeness" of connected customers in high-quality VRP solutions. Let $r_i(j)$ denote the rank of customer j in the sorted distance list of customer i, and we have $r_i(j) = 1$, if j is the closest customer of i. Given a VRP solution, we determine for each customer i the closeness $o_i = \max(r_i(n_1), r_i(n_2))$ as the maximal 'distance rank' of its two connected neighbors n_1 and n_2. If one of the neighbors is the depot, we automatically choose the rank of the other neighbor. By looking at the closeness of all nodes in a high-quality solution, we get an intuition about how many neighbors need to be considered. We performed this analysis on various instances of the instance set of Uchoa et al. (2017), computing and then analysing a solution obtained with the heuristic in Arnold and Sörensen (2017) after 5 minutes of computation time. All of these solutions have a gap of around

1% or less to the best known solutions. The results in Fig. 6 indicate that, for many instances, the connections for 95% of the nodes are with some of the 20 closest neighbors. Furthermore, 99% of the customers have two (or one, if connected to the depot) of their nearest 40 nodes as route neighbors. Only for instances that contain clusters or that have a high variance in demand do we observe a lower closeness. For clustered instances we require long edges that lead out of a cluster, and a high variance in demand results in more chaotic routes with longer edges. These findings are consistent for different instance sizes, which demonstrates that we do not necessarily need to consider more neighbors when facing larger instances.

Moreover, these result suggest that, for most nodes in high-quality solutions, it is sufficient to store information only about the nearest neighbors. Of course, there are exceptions. For instance, we have found good solutions in which some nodes had a very poor closeness of up to 300. However, it is reasonable to assume that we can obtain solutions of similar quality if we exclude those edges since (1) they occur very rarely, and (2) they are relatively long, so it is likely that similar or only slightly worse alternatives are available.

4. Testing and computational Results

4.1. The impact of pruning and limiting stored information

In the previous sections we have described two concepts that enable us to deal with very large scale problems: Heuristic pruning to linearize computational complexity and the limitation of required information to linearize memory usage. In the following we want to validate if and how well these two concepts work.

The idea behind heuristic pruning is to limit the explored neighborhoods of the local search to those moves that are promising. Thus, we should expect lower runtimes if we apply more restrictive pruning strategies. However, at the same time the local search will explore less possible moves, which might prevent it from finding improvements. We should therefore observe a trade-off between runtime and solution quality. If the pruning is too restrictive, the local search is fast but potentially fails to find improving moves, and if the pruning is too lax, the local search might investigate too many alternatives at the cost of significantly higher computation times. In the following experiments we investigate this trade-off. We use the above heuristic to solve the four random instances in Kytöjoki et al. (2007) with 3,000 to 12,000 customers. On each instance, we test different pruning settings, allowing up to 15 minutes of computation time. The CE and EC, both for inter-route optimization, are the most time-consuming operators on those instances, using up to 75% of the total runtime together and, thus, the pruning of these operators is expected to have the highest impact on performance. For both these operators we only consider the c closest neighbors when building and evaluating possible moves. To investigate the effect of pruning on performance, we compare the results of different setups with $c \in \{15, 30, 50, 100\}$.

The results in Fig. 7 reveal that the degree of pruning significantly affects the performance of the heuristic. If we prune too loosely ($c = 100$), we observe severe losses

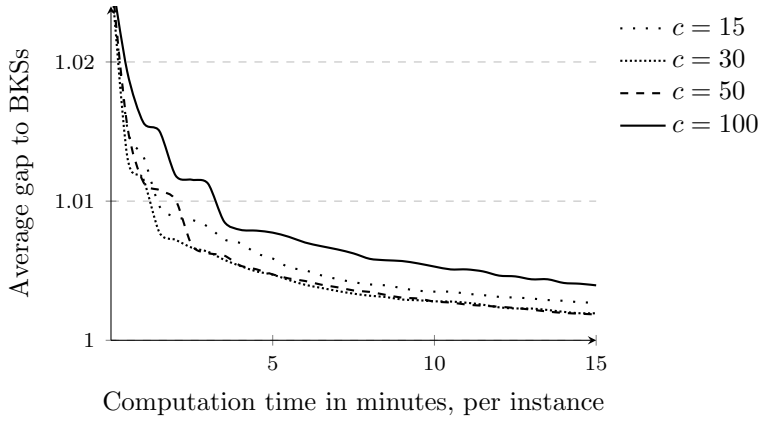

Figure 7: *Average gap (in %) to the best known solutions over time on 4 large-scale instances for different pruning tightness.*

in performance. This can be explained by the additional runtime that is required to search through the larger neighborhoods, so that less iterations can be executed in the same amount of computing time. A looser pruning might still be beneficial in the long run when computing time of several hours is available, since it tend to get stuck in local optima less easily. On the other hand, a more restrictive pruning ($30 \leq c \leq 50$) seems to yield a better performance, and even a very strict pruning ($c = 15$) seems to work well on these very-large scale instance. The gain in computation speed through a tighter pruning seems to compensate for the limited neighborhood size.

The optimal pruning strategy also depends on the nature of the instance. The tested instances have rather short routes, with 15 customers on average, so that the inter-route operators have many options to locate customers in another nearby route. In contrast to that, the waste collection instances described below have few but very long routes of 500 and more customers. With less routes available, it becomes more difficult to change customers into different routes, and we found that a looser pruning ($c = 100$) is required to boost performance. With longer routes, many of a customer's nearest neighbors are in the same route, and less inter-route moves are investigated. In other words, the longer the routes, the less restrictive should the pruning for inter-route optimization be.

In a second set of experiments we investigate the effect of limiting the amount of stored information. We have shown that most customers are connected to relatively close neighbors in high-quality solutions and, thus, it might be sufficient to only store the distance information between close customers. More precisely, instead of the complete distance matrix, we only store the distance between a customer and its closest s neighbors as well as the depot. The distance between any other pair of customers outside of this range is automatically set to infinity. Therefore, none of the local search operators will consider a move that involves such an edge. As a consequence, we indirectly restrict the set of edges that will be taken into consideration and it is therefore necessary to investigate whether this form of pruning comes at the cost of a lower per-

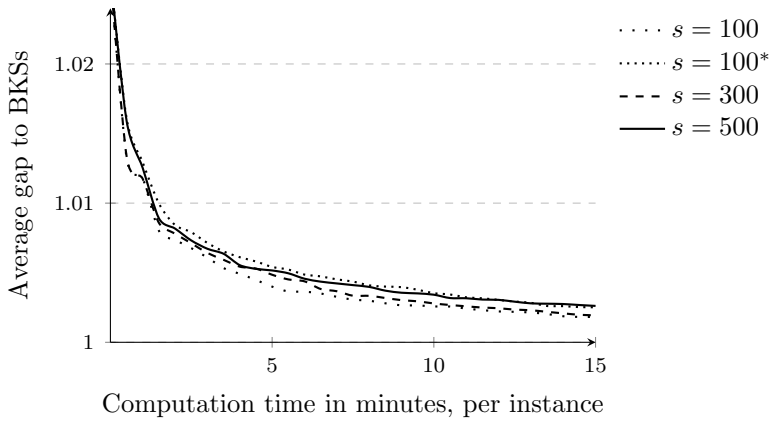

Figure 8: *Average gap (in %) to the best known solutions over time on 4 large-scale instances for different amounts of stored distance values.*

formance. To investigate this trade-off between stored information and solution quality, we perform tests on the same random instances as above, fixing $c = 30$ and comparing $s \in \{100, 300, 500\}$. Finally, we also investigate a setup (marked as 100^*), in which we store the 100 shortest distances prior to the start of the heuristic, but compute all other requested distance values during runtime, as in (Helsgaun, 2000). In other words, in this final strategy distances between further-away nodes are not set to infinity, but computed on request only.

We observe from the results in Fig. 8 that the amount of stored information has only a marginal effect on performance; in other words, the storage of more information, beyond a certain point, does not seem to benefit performance. An explanation for this finding could be that, even though more distance information gives the heuristic more space to work with, it uses this space from time to time to consider longer edges that do not necessarily improve performance in later stages of the search. In other words, too much information is not beneficial and could even potentially harm performance. This result confirms our observations from Fig. 6, and demonstrates that the restriction to important information can reduce memory usage without suffering disadvantages. We want to remark that these tests were carried out on instances with a relatively uniform distribution of customers, and a low variance in customer demand. As a result, customers tend to be connected to near neighbors, as visualized in the right graph in Fig. 9 and the results might not be generalizable to instances with a different configuration. Thus, we will make a compromise and use the 100^* setup in the following for all classes of instances.

4.2. Computational results

We test the performance of the heuristic with the very large scale instances proposed in Kytöjoki et al. (2007). The first group of instances W, E, S and M ($\mathbb{W}$-instances) were extracted from a real-life waste collection application in Eastern Finland. Outgoing from the depot, waste collection trucks need to visit between 7798 and 10,227 specific

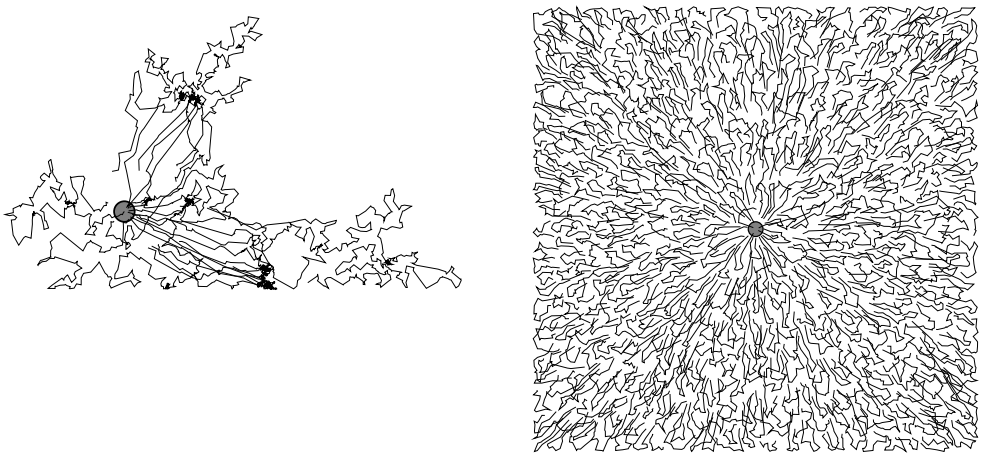

Figure 9: *Solutions for the very large scale instances introduced in Kytöjoki et al. (2007). Waste collection (left), and random placement of customers (right). The edges connected to the depot are not displayed.*

points to pick up waste bins. Since the capacity of the trucks is quite large, one tour can cover 500 and more customers. As a consequence, an efficient intra-route optimization is crucial to successfully solve these instances. At the same time, the depot is quite far away from most customers, so that it is also important to reduce the number of required routes in order to minimize the total distance driven.

The second group of instances contains randomly and uniformly distributed customers that are visited from a centrally located depot ($\mathbb{R}$-instances). Those instances have a size between 3000 and 12,000 customers. In contrast to the $\mathbb{W}$-instances, the capacity is defined in such a way that a route visits 15 customers on average, which is more typical for short-haul problems. Thus, these instances complement the $\mathbb{W}$-instances in the sense that routes are shorter, and the depot is located in the center rather than at the fringe. Both changes affect the problem structure and result in a more diversified instance set. Examples of both instance types are displayed in Fig. 9. For all instances, including the new instances in the next section, the distance between two customers is computed as Euclidean distance, and is not rounded.

We use the heuristic as proposed in Arnold and Sörensen (2017) with the adaptations described in Section 3.2. Chains in the EC operator are limited to two relocations, sequences considered in CE have a maximal length of 100, LK considers moves up to 5-opt ($\mathbb{R}$-instances) and 4-opt ($\mathbb{W}$-instances) among the 10 nearest neighbors, and the initial solution is constructed with CW by only using short edges in the savings list. To optimize efficiency we apply the findings from Section 4.1. For $\mathbb{R}$-instances we only consider the 30 nearest neighbors for inter-route moves (EC and CE), while for the $\mathbb{W}$-instances with long routes we consider the 100 nearest neighbors. For all instances we only store the distance values between a node and its 100 closest neighbors to minimize memory usage, and compute other required distances during runtime.

After the execution of a local search (EC, CE and LK), we penalize a certain edge,

Table 1: *Results on the very large scale instances*

Instance	GVNS			A-G-S (short runtime)			A-G-S (long runtime)		
	Value	Gap	Time	Value	Gap	Time	Value	Gap	Time
W (7,798)	4,559,986	7.37	34.5	4,294,216	1.12	7.8	4,246,802	0.00	39.0
E (9,516)	4,757,566	4.17	83.9	4,639,775	1.59	9.5	4,567,080	0.00	47.5
S (8,454)	3,333,696	3.97	56.2	3,276,189	2.18	8.5	3,206,380	0.00	42.5
M (10,217)	3,170,932	4.35	77.6	3,064,272	0.84	10.2	3,038,828	0,00	51.0
R3 (3,000)	186,220	1.87	4.8	183,184	0.21	3.0	182,808	0.00	15.0
R6 (6,000)	352,702	1.49	24.4	348,225	0.20	6.0	347,533	0.00	30.0
R9 (9,000)	517,443	1.05	57.7	512,530	0.09	9.0	512,051	0.00	45.0
R12 (12,000)	680,833	1.12	108.4	674,732	0.22	12.0	673,260	0.00	60.0
Average		3.17	55.8		0.80	8.3		0.00	41.3

starting with the widest as in Arnold and Sörensen (2017). We perform a global optimization step after $\frac{n}{10}$ moves without improvement; for the $\mathbb{W}$-instance less changes between routes are required since there are relatively few routes and we change the value to $\frac{n}{50}$. If the global optimization did not yield an improvement, we adapt the edge penalization criterion. After we have executed the global optimization and adapted the edge penalization 10 times without any improvements, we reset the search to the previously best found solution and remove all penalties.

Since the depot is far away from the customers on the $\mathbb{W}$-instances, it is also important to minimize the number of routes during the construction of the initial solution. In CW we impose a limit on the number of routes that are constructed, allowing one more route than the (theoretical) minimal number of routes at each point in time. Saving entries that would exceed this limit by creating a new route are ignored but stored for later iterations. After CW we try to remove the smallest route by iteratively trying to relocate all associated customers with the EC-operator.

We compare the performance of our heuristic to the performance of the guided variable neighborhood search (GVNS) introduced in the same paper (Kytöjoki et al., 2007). To get a deeper insight into the trade-off between computation time and solution quality, we provide results after differing running times of the heuristic. For the 'faster' runtime configuration we allow 1 minute per 1,000 customers. For the 'longer' runtime configuration we allow 5 minutes per 1,000 customers. All experiments were executed on a Dell E5550 laptop with an Intel Core i7-5600 CPU working at 2.60GHz with 8 GB memory.

From the results in Table 1 we can conclude that the presented heuristic significantly outperforms the GVNS. On average, the computed solution are more than 3% better, and especially on the $\mathbb{W}$-instances the initial route minimization improves results drastically. Not only have the computed solutions a higher quality, the heuristic also requires less time to obtain them. If we only allow 1 minute per 1,000 customers, the heuristic already improves upon the previous results by more than 2% in less than one sixth of the computation time.

Since there is only one other method to compare with, the question remains whether

Table 2: *Results on the the larger instances in (Uchoa et al., 2017)*

Instance	ILS			HGSADC			A-G-S		
	Value	Gap	Time	Value	Gap	Time	Value	Gap	Time
P90 (801)	74,005.7	0.57	432.6	73,731.0	0.20	289.2	73,587	0.60	4.0
P93 (856)	89,277.6	0.24	153.7	89,238.7	0.20	288.4	89,605	0.61	4.3
P94 (876)	100,417.0	0.70	409.3	99,884.1	0.17	495.4	100,559	0.85	4,4
P95 (895)	54,958.5	1.45	410.2	54,439.8	0.49	321.9	54,889	1.32	4.5
P98 (957)	85,936.6	0.31	311.2	85,822.6	0.18	432.9	86,144	0.55	4.8
P99 (979)	120,253.0	0.89	687.2	119,502.1	0.26	554.0	120,369	0.99	4.9
P100 (1001)	73,985.4	1.71	792.8	72,956.0	0.29	549.0	73,892	1.58	5
Average		0.83	456.6		0.26	418.7		0.93	4.6

the results are rather close to an optimal solution or not. To answer this question, we try to estimate the quality of the computed solutions by extrapolating the results for the largest instances for which we have a good estimate of the optimal solution. More precisely, we test on the instances introduced by Uchoa et al. (2017) (with the same setup as for the $\mathbb{R}$-instances) that have between 800 and 1,000 customers and more than 10 customers per route. As shown in Table 2, the heuristic generates solutions within a 0.93% gap of the best known solutions given 5 minutes of computation time per 1,000 customers. The results are reasonably close to those produced by the ILS, which had a computing time about 100 times larger. Under the assumption that the above heuristic produces solutions of similar quality while scaling linearly in the problem size, the results for the very large scale instances could be estimated to be within a 1% range from the optimal solution. This assumption was shown to hold at least for instances of size 100-1,000 (Arnold and Sörensen, 2017).

In summary, the presented heuristic outperforms the only other heuristic for very large scale instances by a significant margin and computes high-quality, and so far best known, solutions for instances of 10,000 and more customers in less than an hour.

5. New Instances

Most popular benchmark sets for the VRP comprise instances of several hundred customers, and a few include instances with up to 1,000 customers. As a result, newly developed heuristics are exclusively tuned to tackle instances of this size, whereas problems of higher magnitudes are usually entirely neglected. The instances by Kytöjoki et al. (2007) are the only very large scale ones. Given that the instances are a decade old, it is perhaps surprising that our paper is only the second one to tackle them. An explanation could be that, in the race for better performances, researchers have resorted to instances that are well-explored and that enable a more accurate comparison. An additional explanation is that, as we have demonstrated, very large scale problems pose additional challenges that make it more difficult to develop suitable heuristics.

To foster research in the area of very large scale vehicle routing we develop an additional set of very large scale and realistic benchmark instances. These instances are based on parcel distribution in Belgium and, therefore, resemble a real-world problem.

Figure 10: *Solutions for the new instances based on parcel distribution in Belgium: Antwerp (central depot), Ghent (edge depot), and Brussels (edge depot). The edges connected to the depot are not displayed.*

Logistic service providers face the daily task to deliver parcels via vans or trucks from a distribution center to their customers' doorsteps. With the rise of e-commerce in recent years, the number of delivered parcels has grown at a steady pace. In a relatively large city like Brussels (about 1.2 million inhabitants) more than 20,000 parcels have to be delivered every day. With parcel-delivery being a mostly cost-driven business, it is crucial to optimize this distribution. Having fewer delivery vans and fewer delivery kilometers can decrease significantly the costs of service providers, but it also reduces emissions and congestion and thereby increases the quality of life in cities.

On the basis of a dataset containing deliveries in the Flemish and Brussels regions of Belgium, we were able to extract the density of parcel demand for different zones, i.e., the number of delivered parcels per km^2. We use this demand density to sample the location of customers for one delivery day for the cities of Leuven, Antwerp, Ghent and Brussels as well as for the whole region of Flanders. For each of these geographic areas we fix the number of customers and locate them according to demand density. Each customer is assigned a random demand of either 1 (50%), 2 (30%) or 3 (20%) parcels. Furthermore, we consider two delivery scenarios for each of these five customer setups.

In the first scenario, the customers are delivered by vans from a distribution center outside of the city, where the capacity of a van is, depending on the instance, set to 100-200 parcels. In the second scenario, the distribution center is located within the city, and customers are delivered from there via cargo-bikes or electric tricycles that can carry 20-50 parcels at a time. Thus, the routes in the first case are rather long (long-haul) and in the second case rather short (short-haul). We obtain 10 instances that are diverse with respect to customer placement (more clustered versus more evenly distributed), depot location (center versus fringe), and route length (many customers versus few customers).

These instances complement the $\mathbb{W}$ and $\mathbb{R}$-instances in several ways: (1) The instances have a greater variety and also include instances with more clustered customers that are delivered from a central depot, (2) they comprise instances of up to 30,000

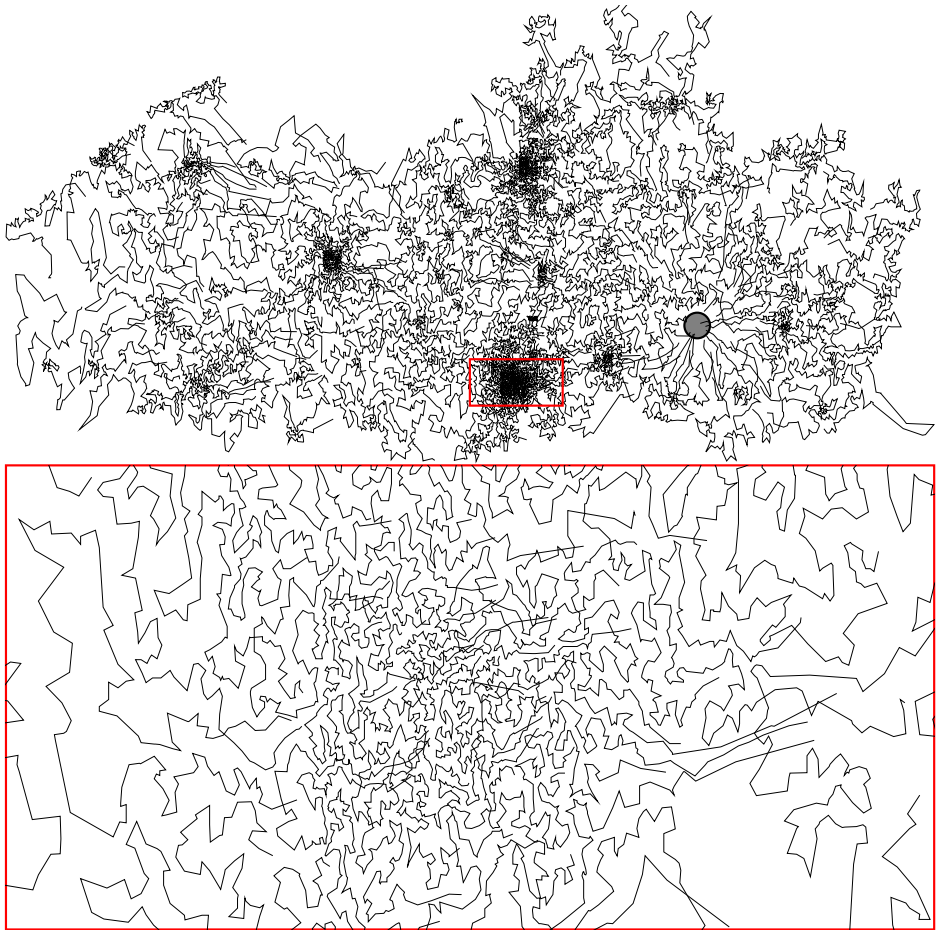

Figure 11: *A solution to distribute parcels within the whole region of Flanders (30,000 customers), with a zoom-in on Brussels.*

customers, and (3) they have more moderate route lengths which are typical for parcel distribution. The new instances are, along with an explanation of the format, publicly available at `http://antor.uantwerpen.be/xxlrouting`, and examples are displayed in Figs. 10 and 11.

For each of the new instances we computed solutions for different computation times to serve as a first benchmark. Hereby, we choose the same setup as for the $\mathbb{R}$-instances. The results are displayed in Table 3. We observe that a well-implemented CW algorithm with a pruned savings-list can already compute reasonable solutions almost instantaneously. Given a few minutes of computation time, we can improve those initial solutions by about 4% on average, especially for instances with longer routes. With more time the improvements become smaller, signifying a convergence of the heuristic.

Table 3: *Results on the new very large scale instances for different computation times T. The depot location (D is either in the center (C) or at the edge (E)), and the average route length (L) vary across instances.*

Instance	D	L	CW			A-G-S (short)			A-G-S (long)		
			Value	Gap	T	Value	Gap	T	Value	Gap	T
L1 (3,000)	C	14.8	200,971	2.94	0.1	195,504	1.36	3.0	195,239	0.00	15.0
L2 (4,000)	E	85.1	125,613	9.39	0.1	117,839	2.62	4.0	114,833	0.00	20.0
A1 (6,000)	C	17.5	498,422	3.06	0.1	484,647	0.22	6.0	483,606	0.00	30.0
A2 (7,000)	E	58.3	322,902	7.85	0.1	304,419	1.68	7.0	299,398	0.00	35.0
G1 (10,000)	C	20.6	490,783	3.0	0.1	478,174	0.35	10.0	476,489	0.00	50.0
G2 (11,000)	E	100	287,371	7.25	0.1	271,350	1.27	11.0	267,935	0.00	55.0
B1 (15,000)	C	29.3	532,558	4.0	0.1	516,042	0.77	15.0	512,089	0.00	75.0
B2 (16,000)	E	87.9	386,048	7.01	0.1	367,561	1.89	16.0	360,760	0.00	80.0
F1 (20,000)	C	29.2	7,525,575	2.78	0.1	7,352,810	0.42	20.0	7,321,847	0.00	100.0
F2 (30,000)	E	117.2	4,805,608	6.16	0.1	4,618,954	2.04	30.0	4,526,789	0.00	150.0
Average				5.34	0.1		1.14	12.2		0.00	61.0

6. Conclusion

In this work we have developed a heuristic that is able to solve VRP instances of several thousands and even tens of thousand of customers within a feasible amount of computing time. We outlined the main challenges of very large scale problems — linearization of time and space complexity — and demonstrated how to tackle them in the context of the VRP using heuristic pruning and a limited information storage. These ideas were embedded in an already very efficient heuristic that uses well-implemented local search operators as its key driver. We further investigated the effect of these pruning techniques, and demonstrated the advantages of a stricter pruning in the context of very large scale problems. The resulting heuristic outperforms the only comparable heuristic by more than 3% in less computation time. Finally, we used real-world data to generate a set of very-large scale instances that reflect problems in parcel distribution to promote more research in this domain.

References

Applegate, D., Cook, W., and Rohe, A. (2003). Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92.

Arnold, F. and Sörensen, K. (2016). Generating problem-specific knowledge (for the vehicle routing problem). *Working paper, University of Antwerp*, 1(1):1.

Arnold, F. and Sörensen, K. (2017). A simple, deterministic, and effcient knowledge-driven heuristic for the vehicle routing problem. *Working paper, University of Antwerp*, 1(1):1.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet management and logistics*, pages 33–56. Springer.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & operations research*, 34(9):2743–2757.

Laporte, G. (2007). What you should know about the vehicle routing problem. *Naval Research Logistics (NRL)*, 54(8):811–819.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Mester, D. and Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34(10):2964–2975.

Reinelt, G. (1991). TSPLIB–A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.

Subramanian, A., Uchoa, E., and Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle routing problem. *Informs Journal on Computing*, 15(4):333–346.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489.

Voudouris, C. and Tsang, E. (1999). Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499.