Computers and Operations Research 107 (2019) 32–42

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Efficiently solving very large-scale routing problems

Florian Arnold [a,*], Michel Gendreau [b], Kenneth Sörensen [a]

[a] University of Antwerp, Departement of Engineering Management, ANT/OR - Operations Research Group, Belgium
[b] Polytechnique Montréal, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Canada

ARTICLE INFO

Article history:
Received 17 May 2018
Revised 13 March 2019
Accepted 13 March 2019
Available online 14 March 2019

Keywords:
Vehicle routing problems
Heuristics
Local search
Large-scale problems

ABSTRACT

The vehicle routing problem is among the most-studied and practically relevant problems in combinatorial optimization. Yet, almost all research thus far has been targeted at solving problems with not more than a few hundred customers. In this paper, we explore how to design a local search heuristic that computes good solutions for very large-scale instances of the capacitated vehicle routing problem in a reasonable computational time. We investigate different ways to reduce the time complexity with pruning and sequential search, as well as the space complexity by restricting the amount of stored information. The resulting algorithm outperforms a previously introduced heuristic for instances of 10,000 and more customers in relatively short computational time. 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 and contain up to 30,000 customers and use our heuristic to solve them.

1. Introduction

Routing problems constitute 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 several characteristics: starting from a source (usually called depot) N 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 of the VRP is provided by Laporte (2007). Due to the complexity of this combinatorial optimization problem, usually only smaller instances of about 200 customers can be solved reliably to optimality by exact methods (Pecin et al., 2017), even though some instances with a specific structure and with up to 600 customers have been solved in Uchoa et al. (2017), given several hours of computational time. 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 computational 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 the focus on instances of this size is unclear. One potential explanation is that it is a result of the size of most popular benchmark instances. The often used instances by Golden et al. (1998) contain up to 483 customers, and the largest problem in the more recent benchmark set by Uchoa et al. (2017) involves 1000 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 with 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 is waste collection, where trucks have to collect waste from tens of thousands of customers. Waste collection is typically modeled as an arc routing problem (Toth and Vigo, 2014; Wøhlk and Laporte, 2018) when the density of customers along a street segment is sufficiently high to consider the street itself as service entity (Assad and Golden, 1995). This condition is usually fulfilled in residential waste collection, whereas in commercial waste collection the customers can be more scattered and it is beneficial to consider the individual nodes (Buhrkal et al., 2012). Another situation where decision-makers would prefer to model waste collection as node routing problem is described in Kytöjoki et al. (2007). With more and more information about,

* Corresponding author.
E-mail address: florian.arnold@uantwerpen.be (F. Arnold).

https://doi.org/10.1016/j.cor.2019.03.006
0305-0548/

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.

As another example, vehicle routing can play a key role in improving the movement of goods in cities as pointed out by Cattaruzza et al. (2017). The authors found, based on the French Surveys on Urban Goods Movement database (Patier and Routhier, 2009), that a large number of deliveries in cities concerns the transportation of small parcels from third-parties on longer delivery routes with 31 and more deliveries per route. Especially the continuing growth of e-commerce results in 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. It is useful to model these problems as node routing problems, if only one or a few customers are delivered on any given street. Larger delivery problems in cities are sometimes tackled by partitioning the geographical region into territories, and assigning each vehicle to a territory. This approach allows to account for the familiarity of the driver with a territory (Zhong et al., 2007), and to keep routing solutions flexible (Janssens et al., 2015). However, the optimization of the delivery routes without prior partitioning can increase the efficiency of the global routing plan, which is especially important given that the delivery business is largely cost-driven.

Consequently, there is a 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, whereas the term '*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 with up to 20,000 customers. Solving such very large instances can also contribute to the generation of new theoretical ideas, since additional obstacles, specifically related to the size of this NP-hard optimization problem, have to be overcome. The literature on the TSP, for example, has covered very large instances since several decades, solving instances with 85,900 nodes and more (Reinelt, 1991). 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.

Efficiently solving instances of this size puts additional requirements on heuristics, both in terms of computational complexity and memory usage. A widely-used approach to reduce computational complexity is *heuristic pruning*. With pruning only promising parts of the solution space are investigated. For instance, the famous heuristic by Lin and Kernighan (1973) only considers local search moves that connect relatively close nodes. Helsgaun (2000) bases the decision of whether to consider a certain edge in a TSP solution on its α-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 TSP solution is a 1-tree, the cost differences in 1-trees provide a good intuition about the likelihood that an edge will be included in an optimal TSP solution. Similar approaches can be used to reduce memory usage. Helsgaun (2000) only stores distance entries of candidate edges computed with the approach above, and all other requested distance values are computed and cached during runtime. Applegate et al. (2003) minimize the representation of large TSPs by building a neighbor graph. Rather than limiting the amount of stored information, it is possible to store information in a highly compressed format as proposed by Kytöjoki et al. (2007). The authors replace distance values by a more condensed representation of speed and twisty-road-curve factors, and the exact distances are recomputed during runtime.

In this paper, we explore how such concepts can be applied in the context of local search to compute good solutions for VRPs with several thousands or more customers in a reasonable amount of computational time. In previous work, we have demonstrated that a well-implemented and well-configured local search is sufficient to compute high-quality solution for VRPs with up to 1000 customers (Arnold and Sörensen, 2019), and the question arising is whether similar ideas can be used to tackle instances of larger magnitudes. On the basis of previous findings about local search for smaller instances (Section 2), we outline the challenges of very large-scale VRPs, and propose corresponding solutions in Section 3. These ideas are tested in various experiments in Section 4 to shed light on the questions 'how complex should local search operators be?', 'how much pruning is necessary?', and 'how much information should be stored?'. As the second main contribution, we generate 10 very large-scale VRP instances in Section 5. 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. Finally, we conclude with a summary of our findings in Section 6. The heuristic, together with benchmark instances, is available at http://antor.uantwerpen.be/routingsolver/.

2. Effective local search for the VRP

Local search has been shown to be one of the few approaches that can successfully tackle a wide range of combinatorial optimization problems (Johnson et al., 1988). It has been particularly successful to solve the TSP (Lin and Kernighan, 1973) and is a major ingredient of many successful heuristics for the VRP (Mester and Bräysy, 2007; Subramanian et al., 2013; Vidal et al., 2012). The fundamental idea of local search is to iteratively improve a solution with small (local) modifications. These modifications are called *moves*, and can either improve a single route (intra-route optimization) or change multiple routes simultaneously in such a way that the overall solution is improved (inter-route optimization). The set of all solutions that can be generated by applying the respective move on a solution s is called the *neighborhood* $\mathcal{N}(s)$ of s. Larger neighborhoods are more likely to contain a solution s^* that improves upon s, but they also exhibit a higher computational complexity since more solutions need to be generated and evaluated.

In Arnold and Sörensen (2019), we investigated this trade-off, and found that larger neighborhoods can yield a good performance if the underlying local search operators are complementary, well-implemented, and effectively pruned. We developed a heuristic that is based on three local search operators. At first, an initial solution is constructed with the heuristic by Clarke and Wright (1964) (CW), and the individual routes are then improved with the heuristic by Lin and Kernighan (1973) (LK). LK has been shown to be a highly efficient heuristic to optimize TSPs and, therefore, represents a fitting operator for intra-route optimization in a VRP. For the optimization between two routes the CROSS-exchange operator (CE) (Taillard et al., 1997) is used, and for the optimization of more than two routes a relocation chain (RC) is proposed, which is based on the concept of the ejection chain introduced in Glover (1996).

These three operators generate complementary and very large neighborhoods, but they can be implemented efficiently if the neighborhoods are not explored exhaustively. Rather than generating and evaluating all possible moves, we determine promising moves with the concepts *sequential search* (Irnich et al., 2006) and *pruning* (Toth and Vigo, 2003).

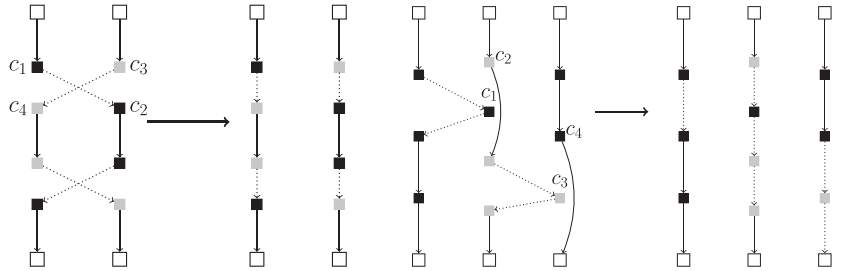

Fig. 1. Illustration of the implemented local search operators for inter-route optimization. (left) A CROSS-exchange move, where substrings of two customers are exchanged between two routes. We only consider the exchange of substrings starting from c_2 and c_4, if the removal and addition of incident edges results in an improvement, i.e., $c(c_1, c_2) + c(c_3, c_4) - c(c_1, c_4) - c(c_3, c_2) \geq 0$. (right) A relocation chain with two subsequent relocations. We only consider a relocation of node c_1 next to its nearest nodes, e.g., c_2.

The idea of sequential search is the deconstruction of complex moves into exchanges of edges. Instead of generating a complete move in one step, existing edges are iteratively exchanged with new edges, and subsequent exchanges are only considered as long as overall improvements are obtained. In the context of LK this concept is also called partial gain criterion. The same idea can be applied to CE, which attempts to exchange two substrings in two different routes. Both substrings have an initial node and a final node, for both of which an edge to a node in the previous route is removed and an edge to a node in the other route is added. It is sufficient to only consider those substrings in which the removal and insertion of the respective edges for the initial nodes improves the solution value, as depicted in Fig. 1. Finally, RC presents a compound move which performs a series of simple relocation moves simultaneously, such that the resulting solution is improved and feasibility is maintained. Starting with a relocation of a customer node from route r_i into route r_j, this relocation can be followed by a relocation of a customer node from route r_j into route r_k (where $i = k$ is possible). Using the idea of sequential search, subsequent relocations are only considered as long as the sum of all previous relocations results in an improved solution.

The size of neighborhoods can be further reduced with heuristic pruning. A common idea is to only consider those moves that involve short edges (Toth and Vigo, 2003), and we applied this type of pruning to all three operators. In the context of LK, only edges between a node and its ten nearest nodes within the same route are considered in moves. Initial nodes of the two substrings in CE can only be connected to their $C = 30$ nearest nodes, and a node in a RC can only be relocated next to one of its $C = 30$ closest nodes. Finally, the best candidate move in the pruned neighborhood is executed, an approach called steepest descent.

At some point, the local search will inevitably reach a local optimum s^* for which the generated neighborhoods $\mathcal{N}(s^*)$ of all three local search operators do not contain a better solution. At this point s^* needs to be perturbed to escape the local optimum, and we proposed the use of a *guided local search* (GLS) (Voudouris and Tsang, 2003). GLS changes the cost evaluation of s^*, rather than s^* itself. A 'bad edge' is determined and penalized by increasing its cost to $c^g(i, j) = c(i, j) + \lambda p(i, j)L$, where $c(i, j)$ indicates the original cost of the edge, $p(i, j)$ counts the number of penalties of edge (i, j), $L = \frac{c(s^{CW})}{N}$ constitutes a proxy for the average cost of an edge in the initial solution s^{CW}, and λ controls the impact of penalties (we always set $\lambda = 0.1$). Previous heuristics proposed to penalize edges with a high *cost* (Mester and Bräysy, 2007). Additionally, we found in a data-mining study that the penalization of wide edges can result in performance gains (Arnold and Sörensen, 2018). 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. For a more formal derivation of this metric, as well as a corresponding experimental study, we refer to the paper above. We condensed these findings in three functions that express the badness of an edge (i, j):

$$b^w(i, j) = \frac{w(i, j)}{1 + p(i, j)} \qquad b^c(i, j) = \frac{c(i, j)}{1 + p(i, j)}$$
$$b^{w,c}(i, j) = \frac{w(i, j) + c(i, j)}{1 + p(i, j)} \tag{1}$$

After each perturbation phase we change the applied badness function in a deterministic manner, from $b^w(\cdot)$ to $b^c(\cdot)$, from $b^c(\cdot)$ to $b^{w,c}(\cdot)$, from $b^{w,c}(\cdot)$ to $b^w(\cdot)$ and so forth. The worst edge according to the current badness function is penalized, and the local search operators CE and RC subsequently attempt to remove it. Moves that improve the solution under $c^g(\cdot)$ are considered as candidate moves, and the best candidate move is executed. Thus, with more received penalties it becomes more likely to find a move that removes a certain edge. After a sufficient number of moves have been executed under $c^g(\cdot)$, we obtain a perturbed solution s^p. We found that $P \leq 100$ moves are sufficient to effectively perturb the solution, and in the remainder of this paper we use $P = 100$. Note that s^p might constitute a worse solution than the previous local optimum s^*, since a move that improves the solution under $c^g(\cdot)$ does not necessarily improve the solution under $c(\cdot)$.

The perturbed solution is then improved with the local search operators, using the standard evaluation $c(\cdot)$. In this optimization step, first CE and then RC are iteratively applied on the perturbed solution, and each found inter-route move is immediately followed by an intra-route optimization of the involved routes with LK. If in one iteration neither CE nor RC find an improving move, a local optimum is reached and the corresponding solution is perturbed. In summary, we obtained the heuristic outlined in Algorithm 1, which we denote as knowledge-guided local search (KGLS) in the following.

KGLS is entirely based on a local search with the steepest descent acceptance criterion, and thus, it works in a deterministic fashion. We observed that it computes high-quality solutions on a wide range of benchmark instances in short computational times. For most instances with up to 1000 customers, it does not require more than a few minutes to compute a solution with a gap of 1% and less to the best known solutions, as demonstrated in Fig. 2 (we want to remark that gaps between 0.5% and 1% may be considered large gaps from a theoretical point of view, but are perfectly acceptable in many real applications). This effective time-quality trade-off can be attributed to a good scalability. In the following, we investigate how this scalability is a suitable basis to extend the heuristic to solve instances of larger magnitudes.

3. From large VRPs to very large VRPs

Solving very large instances with several thousands or tens of thousands of customers poses additional challenges to heuristics.

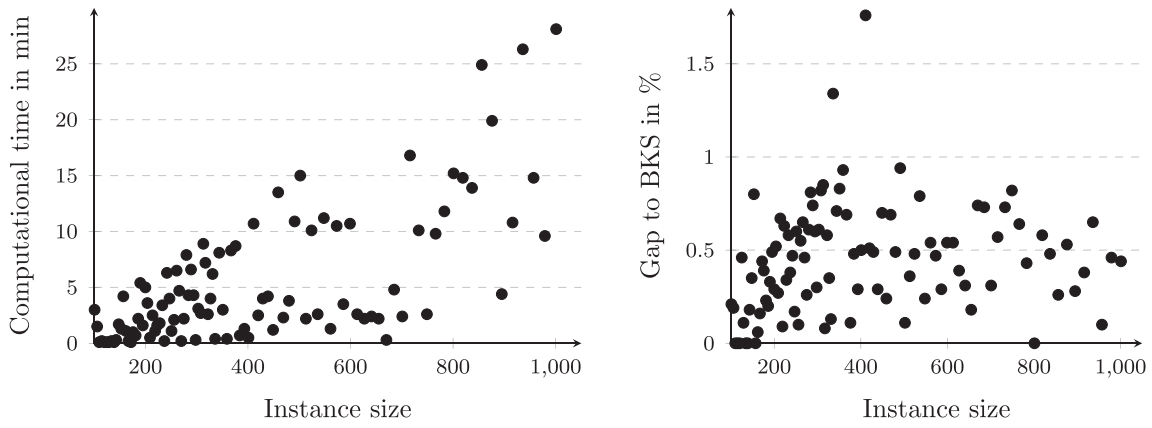

Fig. 2. Computational time (left) versus performance (right), expressed as gap to the best known solutions (BKS), in dependence of instance size on the 100 instances by Uchoa et al. (2017). KGLS computes solutions of a similar high quality within computational times that grow approximately linear in the instance size.

Algorithm 1 Knowledge-guided local search heuristic (KGLS), as developed in Arnold and Sörensen (2019).

Construction. Construct an initial solution with the heuristic by Clarke and Wright (1964). Optimize the individual routes with LK.
Initial optimization. Iteratively apply CE and RC on initial solution until a local optimum is reached. Whenever a route is changed, re-optimize it with LK. Set $b(\cdot) = b^w(\cdot)$.
while time limit not reached **do**
  Perturbation.
  while not $P = 100$ moves have been made **do**
    Penalize edge (i, j) with the highest value $b(i, j)$ by incrementing $p(i, j)$.
    Apply CE and RC on (i, j), using $c^g(\cdot)$ as evaluation criterion.
  end while
  Apply LK on all routes that were changed during perturbation. Change $b(\cdot)$.
  Optimization. Iteratively apply CE and RC on all routes that were changed
    during perturbation until a local optimum is reached ($c(\cdot)$ as evaluation
    criterion). Whenever a route is changed, re-optimize it with LK.
end while

Firstly, the computational complexity increases rapidly as a function of the problem size. In Fig. 3 we plot the computational times of two state-of-the-art heuristics, the hybrid genetic heuristic (HGSADC) by Vidal et al. (2012) and the iterated local search (ILS) by Subramanian et al. (2013) against problem size, 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 computational time. Even though these observations have to be treated carefully, since the runtime of HGSADC and ILS could probably be reduced by decreasing the termination criterion or removing some neighborhoods for larger instances without suffering a large loss in solution quality, the central statement is that very large-scale VRPs demand heuristics to scale well in problem size. KGLS appears to scale well for instances with up to 1000 customers, but care must be taken to reduce the size of the considered neighborhoods as much as possible.

Secondly, the amount of memory used (RAM) might present an obstacle for very large instances. For instance, storing the complete distance matrix between each pair of nodes requires N^2 entries, where N is the number of nodes, and if a distance entry is represented by a float (4 bytes), the complete matrix for a 20,000 customer instance would require about 1.5GB memory. Additionally to the distance matrix, information about penalized cost values and received penalties per edge need to be stored, a solution needs to be encoded and candidate local search moves have to be saved. All of this data might increase memory usage beyond available memory.

Both of these issues can be addressed by ignoring parts of the solution space. For instance, it is generally assumed that long edges do or should not appear in high quality solutions and, thus, moves that contain such edges can simply be ignored. Going one step further, it might not even be necessary to store information about such edges. If distance entries between far-away nodes are never retrieved because the corresponding edges will not be taken into consideration, then there is no need to store their value. In short, we can drastically reduce computational complexity and the amount of used memory by asking the question 'Which edges should be considered during runtime?'.

We approach 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 which contains the distances to all other customers. If j is the closest customer of i, we define $r_i(j) = 1$. 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 analyzing a solution obtained with KGLS. All of these solutions have a gap of around 1% or less to the best known solutions.

The results in Fig. 4 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 50 nodes as route neighbors. Only for instances that contain clusters or that have a high variance in demand we do frequently observe edges that are formed between further away nodes. Instances with customer clusters require long edges that lead out of a cluster, and a high variance in demand complicates the bin-packing problem of assigning customers to routes, which results in more chaotic routes with longer edges. These findings are consistent for different instance

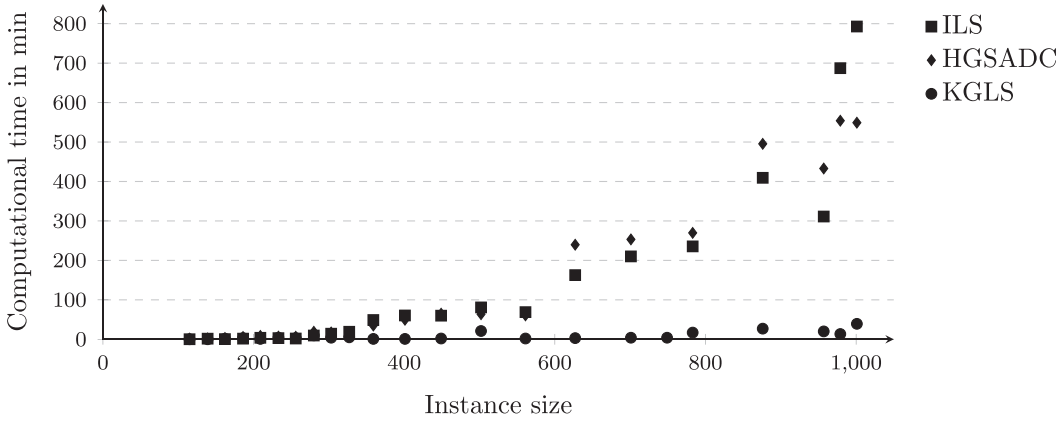

Fig. 3. Computational time (normalized to an Intel Xeon CPU with 3.07 GHz) as a function of instance size on the instances of Uchoa et al. (2017) for two state-of-the-art heuristics from the literature (ILS and HGSADC) and KGLS. The data shows that the runtime of both heuristics grows in a nonlinear fashion, and much faster than the runtime of KGLS (we want to remark that ILS and HGSADC were not configured to achieve minimal runtime, but rather maximal quality). All three heuristics produce high-quality solutions on these instances with average gaps ILS 0.52%, HGSADC 0.19%, KGLS 0.44%.

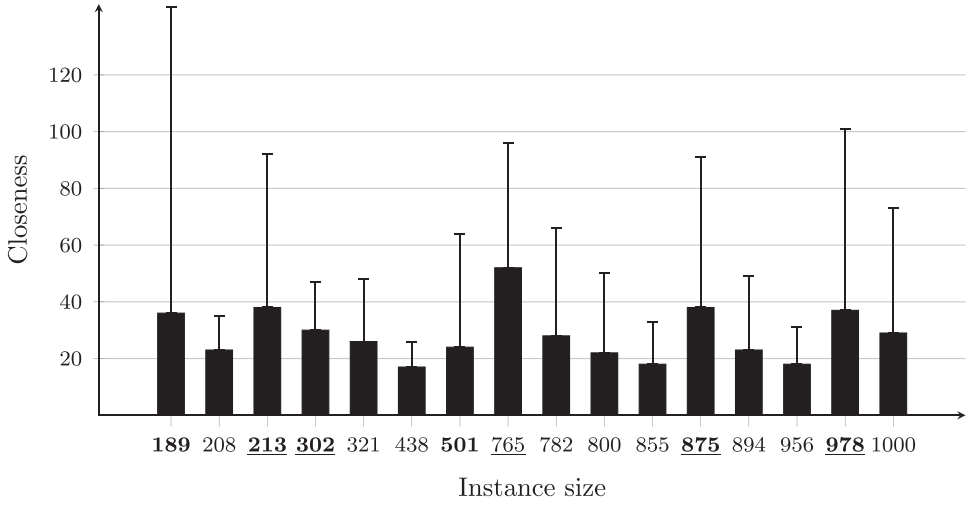

Fig. 4. Maximal closeness of 95% of the customers (bars), as well as maximal closeness of 99% of the customers (error bars) for different instances. Instances with a high variety in demand (1–100) are underlined, instances with clusters are highlighted in bold.

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 about the nearest neighbors. Of course, there are exceptions, and we also found good solutions in which some nodes had a very poor closeness of up to 300. However, it is reasonable to assume that one can obtain solutions of similar quality if those edges are excluded, 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. Effective local search for very large-scale VRPs

In the following, we use the observations from the previous section to investigate the functioning of local search for very large-scale instances. We hypothesize that an effective local search on larger instances requires a more drastic reduction of the considered neighborhoods. The size of neighborhoods can be reduced by using less complex local search operators, and by pruning the neighborhoods of those operators more effectively. Finally, we investigate in how far information storage impacts algorithmic performance.

We perform all experiments on the eight instances introduced by Kytöjoki et al. (2007). The first group of instances ($\mathbb{R}$-instances) contains randomly and uniformly distributed customers that are visited from a centrally located depot. Those instances have a size between 3000 and 12,000 customers. The vehicle capacity is defined in such a way that a route can visit about 15 customers, which is more typical for short-haul problems. The second group of instances ($\mathbb{W}$-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 points to pick up waste bins. The capacity of the trucks is quite large, and one route can cover 500 and more customers. Therefore, the large number of rather short routes in the $\mathbb{R}$-instances requires a heavy inter-route optimization, whereas the long routes in the $\mathbb{W}$-instances put a stronger focus on intra-route optimization. Examples of both instance types are displayed in Fig. 5.

We test different configurations of KGLS to solve each instance. The performance of a certain configuration is expressed as the average gap between the best solutions found within a specific time and the best known results, as reported by Kytöjoki et al. (2007). KGLS has been implemented in Java and all tests have been performed within the Eclipse development

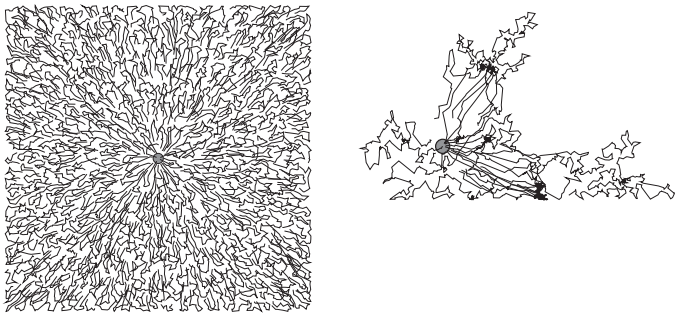

Fig. 5. Solutions for the very large-scale instances introduced in Kytöjoki et al. (2007). Random placement of customers (left) and waste collection (right). For a better visibility, the edges connected to the depot are not displayed.

environment on an AMD Ryzen 3 1300X CPU working at 3.5 GHz on Windows 10, using a single thread. According to PassMark Software (2018) this setup is about 4 times faster than the setup used in Kytöjoki et al. (2007) (AMD Athlon64 3000+).

4.1. Construction of an initial solution

KGLS uses CW to construct an initial solution for a VRP instance. The route computation in CW is based on a savings list that contains, for each pair of nodes, the savings that could be obtained when connecting the respective nodes. As a consequence, the savings list and the number of computational steps grow quadratically in the instance size, which might render this standard version inefficient for larger problems. We have observed in the previous section that nodes in high-quality solutions are typically connected to nearby nodes, and thus conjecture that it is sufficient to limit the savings list to savings between each node and its 100 nearest nodes, thereby linearizing the length of the list and the number of computational steps. Another construction heuristic with a linear number of steps is the famous sweep heuristic. This heuristic transforms the Cartesian coordinates of nodes into polar coordinates (described by angular and radial coordinates), where the depot is the center of the polar coordinate system. The nodes are then sorted according to their angular coordinates, and each node is iteratively connected to the previously unconnected node with the next larger angular coordinate. This process is iterated until the capacity limit is reached, in which case a new route is started. After all customers have been assigned to routes, each route is optimized in itself. As a result, the customers are bundled in geographic corridors, which resembles the idea of districting we mentioned in the introduction.

In the following, we compare the suitability of these three construction heuristics CW, CW^{100} and *Sweep* for very large instances. *Sweep* requires a post-optimization step in which the routes are improved, and to allow for a fair comparison we apply intra-route optimization with LK on all constructed initial solutions. Additionally, the computational time required to read and store information about the instance and distances (between 2 s for R3 and 30 s for R12) is added to the overall time.

From the results in Table 1 we observe that CW is a powerful heuristic for very large instances, and can compute reasonable solutions within seconds. However, the size of the entire savings list can quickly exceed the available heap space (in our case 4GB), which becomes a problem when facing even larger instances with 20,000 or 30,000 customers. This problem can be circumvented with CW^{100}, which only considers the savings between a node and its 100 nearest neighbors. This version also speeds up the construction process while the solution quality is only marginally affected. Only for some $\mathbb{W}$-instances CW^{100} does not sufficiently minimize the numbers of routes (which is important on those instances where the depot is far away from the customers), resulting in larger gaps. In contrast to that, it appears that *Sweep* does not compute good initial solutions for the $\mathbb{R}$-instances with many short routes. For those instance types one can certainly find better districting strategies, while for instances with long routes *Sweep* performs decently. Overall, CW^{100} appears to be a well-scaling construction heuristic for very large instances, and we will use it for all instances except for the $\mathbb{W}$-instances for which we will use the standard version of CW to obtain initial solutions with fewer routes.

4.2. Complexity of operators

We have observed that a local search composed of LK, CE and RC functions effectively for instances with up to 1000 customers. However, the inherent complexity of these operators might be too large when facing instances of greater magnitude. This raises the question whether a local search needs to be adapted to successfully solve very large-scale instances.

LK 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; Helsgaun, 2000). Since LK exhibits a non-linear time complexity, the scaling of LK in the context of VRPs is determined by the length of routes. An important lever to adjust the runtime behavior of LK is the maximum number κ of edge exchanges per move. CE exchanges substrings in two different routes, and more combination of substrings have to be considered if routes increase in length. This complexity can be tackled by imposing a limit on the length of considered substrings, as suggested by Taillard et al. (1997). We found that the preceding sequential search already sufficiently reduces the neighborhood, and a restriction on the length of substrings does not yield performance gains in our implementation of CE. Finally, we

Table 1
Results of different construction heuristics on the very large-scale instances by Kytöjoki et al. (2007) (*N* denotes the number of nodes). The gaps are reported in % to the results in the same paper, along with the time *T* in seconds.

Instance (*N*)	CW			CW^{100}			*Sweep*		
	Value*	Gap	*T*	Value	Gap	*T*	Value	Gap	*T*
W (7,798)	4,600,457	0.89	79	4,594,582	0.76	19	4,929,615	8.11	34
E (9,516)	4,758,438	0.02	103	4,771,803	0.30	31	5,100,387	7.21	54
S (8,454)	3,370,304	1.10	85	3,467,753	4.02	22	3,650,841	9.51	37
M (10,217)	3,162,326	-0.27	136	3,396,401	7.11	30	3,360,857	5.99	51
R3 (3,000)	188,650	1.30	8	188,523	1.24	3	232,212	24.70	5
R6 (6,000)	355,361	0.75	31	355,481	0.79	9	462,084	31.01	11
R9 (9,000)	521,311	0.75	77	521,646	0.81	18	696,312	34.57	20
R12 (12,000)	–	–	–	685,975	0.76	34	931,042	36.75	33

*For some instances, we could not compute a solution with a heap space of 4GB.

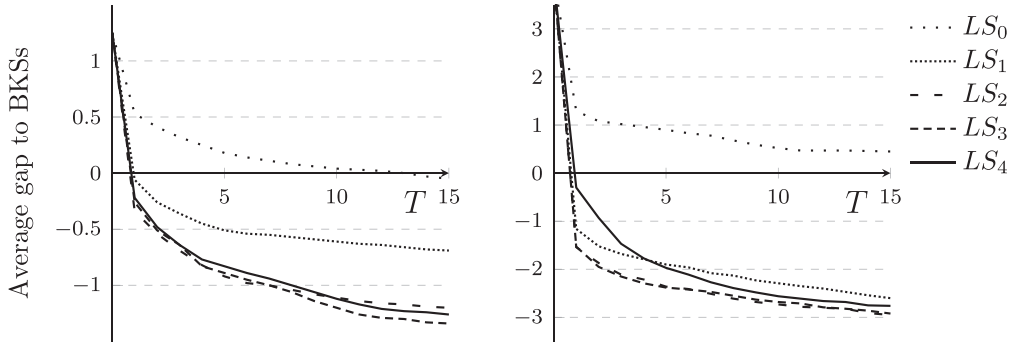

Fig. 6. Average gap (in %) to the best known solutions (BKSs) over time (T in minutes) on the $\mathbb{R}$-instances (left) and $\mathbb{W}$-instances (right) for different local search configurations.

Table 2
Local search operators used in the different setups.

	Intra-route operators	Inter-route operators
LS_0	2-opt	relocate, swap, crossover
LS_1	LK ($\kappa = 4$)	CE
LS_2	LK ($\kappa = 4$)	CE, RC ($\rho = 2$)
LS_3	LK ($\kappa = 4$)	CE, RC ($\rho = 3$)
LS_4	LK ($\kappa = 5$)	CE, RC ($\rho = 3$)

observed that RC requires the highest runtime among the three operators, while the considered neighborhood grows rapidly in the number of subsequent relocations. Therefore, it is sensible to define an upper bound ρ for the number of combined relocations. For smaller instances we found $\rho = 3$ to be a good choice.

On the basis of these ideas, we compare the performances of the different local search configurations in Table 2 within KGLS. LS_0 contains simple neighborhoods that can be readily implemented. The move *relocation* changes the route of one customer, the move *swap* exchanges the route positions of two customers, and *crossover* exchanges two substrings of customers between different routes, where both substrings are connected to the depot (a special case of a CROSS-exchange with only two edge exchanges). LS_1 generates larger neighborhoods with LK and CE, LS_2 additionally includes the relocation chain operator, which is further extended in LS_3 to three simultaneous relocations. Finally, LS_4 allows for a more complex intra-route optimization with up to five edge exchanges. In all configurations, we keep the degree of inter-route pruning ($C = 30$) as well as the memory usage (the 100 shortest distances for each node are computed prior to the start of the heuristic, and all other requested distance values are computed during runtime, see Section 4.4) fixed.

The experimental results reported in Fig. 6 demonstrate that the more complex local search operators work surprisingly well on both benchmark sets, and each configuration containing LK and CE quickly improves upon the best known solution of each instance. On the $\mathbb{R}$-instances we observe that all configurations with a more complex inter-route optimization perform similarly well, with LS_3 computing the best results within 15 min. This setup is also the best configuration on the $\mathbb{W}$-instances, while an increase in the complexity of intra-route optimization appears to slightly diminish performance. Overall, complex intra-route operators appear to work well on these very large instances, while care should be taken to not dedicate too much computational time for intra-route optimization.

4.3. Pruning

Heuristic pruning attempts to limit the explored neighborhoods by only considering promising moves. The more restrictive the respective pruning strategies are, the faster the neighborhoods can be explored, while, on the other hand, the local search explores less moves, which might prevent it from finding improvements. The result is a possible trade-off between runtime and solution quality.

In the following experiments we investigate this trade-off with the best local search configuration LS_3. The inter-route operators CE and RC only consider moves in which a node is connected to one of its C closest nodes, and we investigate the effect of pruning on performance by varying $C \in \{10, 20, 30, 50\}$. On the basis of the results in Fig. 4, we would expect that a more restrictive pruning can be beneficial, since edges in high-quality solutions are typically between spatially close nodes. Instead of considering the C nearest nodes in terms of cost $c(\cdot)$, one could also define nearness with a different metric. A good example is the α-nearness defined in Helsgaun (2000) for the TSP. The α-nearness $c^{\alpha}(i, j)$ between two nodes i and j is defined as the cost difference between the optimal 1-tree and the minimum 1-tree that contains edge (i, j). Since every TSP solution 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. We implemented this idea for the VRP, and construct the 1-trees out of all customer nodes.

The results in Fig. 7 indicate that a restrictive pruning ($C \leq 20$) appears to be effective on the $\mathbb{R}$-instances, with $C = 20$ yielding the best performance (on smaller instances with up to 1000 customers we obtained the best results with $C = 30$). The gain in computation speed through a tighter pruning seems to compensate for the reduction in neighborhood size, so that more iterations can be executed in the same computational time. On the other hand, for the $\mathbb{W}$-instances a looser inter-route pruning ($C = 50$) appears to be the best choice. This difference could be explained by the occurrence of customer clusters, which require longer edges to enter and exit the clusters. Additionally, due to the long routes many of a customer's nearest neighbors are likely to be in the same route and cannot be used as the starting point for inter-route moves.

These characteristic differences between the two instance types are also reflected in the results with $c^{\alpha}(\cdot)$. For reasons of clarity, we only plot the performance curves of the best configurations ($C^{\alpha}_{best} = 20$ for the $\mathbb{R}$-instances and $C^{\alpha}_{best} = 50$ for the $\mathbb{W}$-instances). Whereas on the $\mathbb{R}$-instances with evenly distributed customers and relatively short routes the 'pure' distance seems to be a good metric to filter promising moves, the α-nearness appears to be the more effective metric on instances with long routes and customer clusters. We want to remark that the absolute differences in performance are rather small because both metrics compute similar sets of nearest neighbors. For instance, for $C = C^{\alpha} = 20$ we observed that 86% of the nearest nodes in terms of $c(\cdot)$ are also nearest nodes in terms of $c^{\alpha}(\cdot)$.

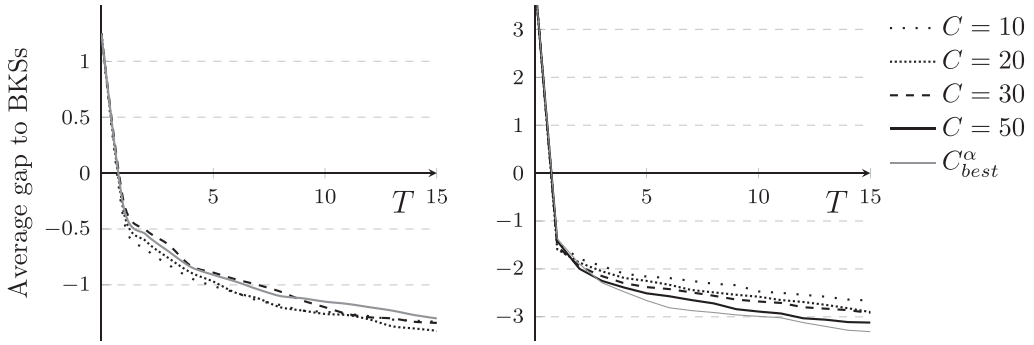

Fig. 7. Average gap (in %) to the best known solutions (BKSs) over time (T in minutes) on the $\mathbb{R}$-instances (left) and $\mathbb{W}$-instances (right) for different pruning strategies. For the larger $\mathbb{R}$-instances, the inital optimization for $C = 50$ requires too much time and the corresponding results are excluded.

4.4. Memory usage

In a last set of experiments we investigate the effect of limiting the amount of stored information. Instead of storing the complete distance matrix, only the distance between a customer and its S closest nodes as well as the distance to the depot is stored. The distance between any other pair of customers outside of this range is automatically set to infinity. Since none of the local search operators will consider a move that involves such an edge, the set of edges that will be taken into consideration is thereby indirectly restricted. Note that the determination of the S nearest neighbors per customer requires the computation and sorting of possibly all distance values. The tuples $< neighbor, distance\_value >$ for the S nearest neighbors are then stored in a hashmap.

We compare the performance for $S \in \{100, 500\}$, keeping the pruning $C = 20$ ($\mathbb{R}$) and $C = 50$ ($\mathbb{W}$), and the best local search configuration LS_3 fixed. Since it was not possible to store the whole distance matrix for some instances (KGLS already requires up to 2.8GB of RAM during runtime for $S = 500$), we restrict the maximum storage to $S = 500$. Additionally, we also investigate a setup marked as 100*, in which the 100 shortest distances are stored prior to the start of the heuristic, while all other requested distance values are computed during runtime as in Helsgaun (2000). If the evaluation of a local search move requires a distance value that is not included in the hashmap, the Euclidean distance between the respective nodes is computed once without storing it. In other words, in this strategy distances between further-away nodes are not set to infinity, but computed on request only. To test whether the storage of some information is beneficial, we also investigate the setup 0*, in which all distance values are computed on request and none are stored.

We observe from the results in Fig. 8 that the storage of more information, beyond a certain point, does not seem to benefit performance on the $\mathbb{R}$-instances. A strict pruning with $S = 100$ and $S = 500$ even deteriorates the performance on the $\mathbb{W}$-instances, for which the consideration of long edges is important. Finally, it appears beneficial to pre-process and store at least the most frequently requested distance values since $S = 100^*$ dominates $S = 0^*$ on both instance sets. In summary, the storage of often-requested information about short edges, complemented with a computation of the remaining edge information on request appears to be a good approach to tackle the memory challenges of very large instances.

4.5. An effective configuration for very large instances

On the basis of the above findings, we obtain the following configuration (which we call $KGLS^{XXL}$) for very large instances. The initial solution is constructed with CW^{100} by only considering savings between a customer and its 100 nearest nodes. The $\mathbb{W}$-instances with very long routes constitute an exception for which we resort to the standard CW version. Memory usage is reduced by storing the distance values between a customer and its 100 nearest neighbors, while all other distance values are computed on request during runtime. We use the local search configuration LS_3 with relatively complex inter-route operators, and a slightly less complex intra-route operator. The intra-route operators are pruned more restrictively ($C = 20$) for VRP instances with shorter routes, while for instances with very long routes a looser pruning ($C = 50$) appears beneficial.

We test $KGLS^{XXL}$ on the $\mathbb{R}$-instances and the $\mathbb{W}$-instances for differing running times, and compare the performance to the results of the guided variable neighborhood search (GVNS) introduced by Kytöjoki et al. (2007). In the 'short' runtime setup we allow the same computational time as that used by the GVNS, given that our machine is about 4 times faster. In the 'long' runtime setup we allow 5 min per 1000 customers. From the results in Table 3 we observe that $KGLS^{XXL}$ outperforms the GVNS and improves the results on each instance in less than an hour. These

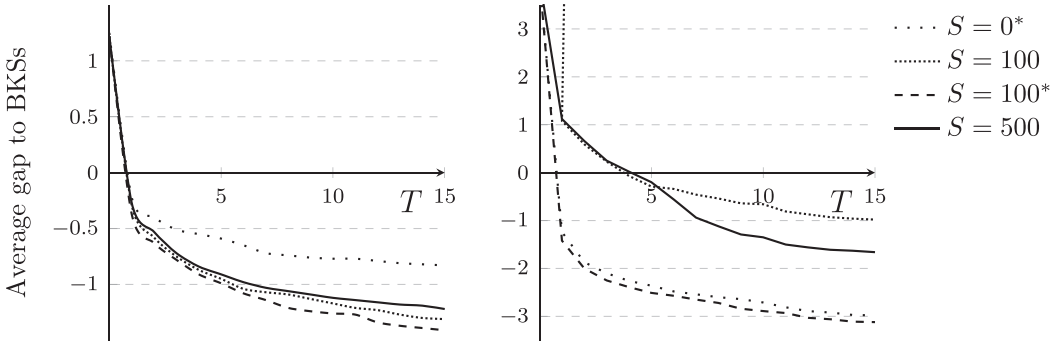

Fig. 8. Average gap (in %) to the best known solutions (BKSs) over time (T in minutes) on the $\mathbb{R}$-instances (left) and $\mathbb{W}$-instances (right) for memory usage strategies.

Table 3
Results of the presented heuristic on the very large-scale instances by Kytöjoki et al. (2007), the instance size is given in parenthesis. The gaps are reported in % to the results in the same paper, given different computational times (T in minutes).

Instance (N)	GVNS		KGLS^{XXL} (short runtime)			KGLS^{XXL} (long runtime)		
	Value	T^*	Value	Gap	T	Value	Gap	T
W (7,798)	4,559,986	34.5	4,498,958	−1.34	8.6	4,481,423	−1.72	39.0
E (9,516)	4,757,566	83.9	4,522,940	−4.93	21.0	4,507,948	−5.25	47.5
S (8,454)	3,333,696	56.2	3,204,376	−3.88	14.1	3,189,850	−4.31	42.5
M (10,217)	3,170,932	77.6	3,095,119	−2.39	19.4	3,071,090	−3.15	51.0
R3 (3,000)	186,220	4.8	183,113	−1.67	1.2	182,206	−2.16	15.0
R6 (6,000)	352,702	24.4	347,872	−1.37	6.1	347,224	−1.55	30.0
R9 (9,000)	517,443	57.7	511,656	−1.12	14.4	511,378	−1.17	45.0
R12 (12,000)	680,833	108.4	672,870	−1.17	27.1	672,456	−1.23	60.0
Average		55.8		−2.23	14.0		−2.57	41.3

*Based onPassMark Software (2018), the used CPU is about 4 times slower.

Table 4
Performance statistics for each local search operator on four instances with different size N: The number of evaluated moves per second Ev./s, the number of executed moves per second Ex./s, and the proportion of runtime required by each operator $\%T$.

N	LK			CE			RC		
	Ev./s	Ex./s	$\%T$	Ev./s	Ex./s	$\%T$	Ev./s	Ex./s	$\%T$
138	539,895	370	37%	205,782	581	26%	4,129,918	497	37%
560	389,575	131	26%	137,611	257	26%	3,192,632	241	48%
3000	104,638	37	9%	98,043	87	39%	4,905,093	80	52%
9000	17,415	3	2%	95,911	29	49%	6,356,900	7	49%

results suggest that a local search with larger neighborhoods, combined with a fitting pruning and memory strategy, works effectively for very large instances.

4.6. Analysis of the local search configuration

In the following, we investigate the behavior of KGLS^{XXL} more accurately and explore how many and which local search moves are evaluated and executed during runtime. A move is defined to be 'evaluated' if all involved cost computations have been completed. Note that due to sequential search and preceding feasibility checks even more moves might be evaluated partially. We conduct this analysis for the smaller instances X-n139-k10 and X-n561-k42 of Uchoa et al. (2017) as well as for the larger instances R3 and R9 of Kytöjoki et al. (2007). All of these instances share many characteristics (randomly distributed customers are delivered from a central depot on routes that visit around 15 customers), which should allow to investigate the impact of instance size on the behavior of the local search.

Table 4 displays the average number of evaluated and executed moves per second, as well as the proportion of time that is required by each operator. We observe that the number of evaluated inter-route moves does not seem to depend on instance size, and on an instance with 9000 customers the local search evaluates almost as many CE and RC moves as on an instance with 138 customers. With growing instance size there are more routes of the same length, and thus, less time is proportionally spent on intra-route optimization which results in fewer LK evaluations per second. Note that the number of evaluations per time unit spent on LK remains almost constant. On the other hand, we observe that the number of executed moves per second decrease with larger instances. This observation can be attributed to the larger neighborhoods that occur in larger instances, and since the evaluation of larger neighborhoods requires more time, less moves can be executed overall.

In Table 5 we investigate in more detail which moves are executed during runtime. Hereby, *swap* denotes the exchange of two customers, while *CROSS-ex.* denotes the exchange of longer substrings between two routes. RC performs ρ relocations simultaneously. As observed above, the proportion of intra-route moves (LK) decreases with the instance size, where the less complex moves 2-opt and 3-opt are executed more frequently than a complex 4-opt move. In the context of inter-route moves, we observe that the simple swap move gains more importance with growing instance size, and constitutes almost 48% of all executed moves for $N = 9000$. On the other hand, we observe that the complex moves CROSS-ex. and RC remain important even on very large instances. Notably, a single relocation is executed much less frequently than multiple relocations in parallel, which highlights the benefits of the RC operator. These findings appear to be consistent for different instance sizes.

5. New Instances

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. Logistics 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, and in a relatively large city like Brussels 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 the given demand

Table 5
Number of executed moves per second (rounded) for different move types. The overall proportion of the respective move is given in parenthesis.

N	LK			CE			RC	
	2-opt	3-opt	4-opt	relocation*	swap	CROSS-ex.	$\rho = 2$	$\rho = 3$
138	177 (12.2%)	137 (9.5%)	59 (3.9%)	17 (1.2%)	165 (11.4%)	407 (28.1%)	92 (6.3%)	397 (27.5%)
560	69 (11.0%)	47 (7.4%)	15 (2.4%)	33 (5.3%)	87 (13.8%)	154 (24.5%)	46 (7.4%)	177 (28.2%)
3000	20 (9.6%)	13 (6.3%)	4 (2.2%)	2 (1.1%)	55 (27.1%)	31 (15.1%)	35 (17.2%)	44 (21.5%)
9000	1 (3.4%)	1 (2.4%)	0.4 (1.0%)	3 (8.7%)	19 (48.0%)	9 (23.0%)	5 (11.4%)	1 (2.0%)

*Also includes RC moves with $\rho = 1$.

Fig. 9. Solutions for the new instances based on parcel distribution in Belgium: Antwerp (central depot), Ghent (eccentric depot), and Brussels (eccentric depot). The edges connected to the depot are not displayed.

density. Each customer is assigned a demand of either one, two or three parcels with probability 50%, 30% and 20%, respectively. Furthermore, we consider two delivery scenarios for each of these five regions.

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 and in the second case rather short. We obtain 10 instances that are diverse with respect to customer placement (more clustered versus more evenly distributed), depot location (central versus eccentric), and route length (many customers per route versus few customers per route). Solutions for instances with an eccentric depot location tend to have longer edges between the depot and the first and the last customer of a route, and thereby differ from instances with a central depot location (Fig. 9 visualizes some examples). All distances are Euclidean and rounded to the nearest integer.

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 with up to 30,000 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 http://vrp.atd-lab.inf.puc-rio.br, and examples are displayed in Fig. 9 and Fig. 10.

For each of the new instances we compute solutions with KGLS^{XXL} to serve as a first benchmark, given 3 min (short runtime) and 12 min (longer runtime) of computational time per 1000 customers, with LS_3 as local search, $C = 20$ for intra-route pruning and $S = 100^*$ to handle memory. The results are displayed in Table 6. We observe that a well-implemented CW algorithm with a pruned savings-list computes reasonable solutions almost instantaneously.

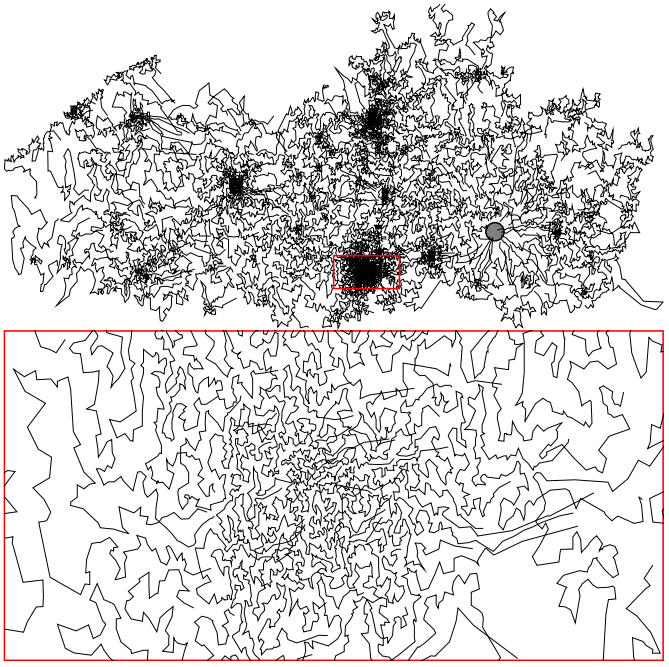

Fig. 10. A solution to distribute parcels within the whole region of Flanders (Instance F2 - 30,000 customers), with a zoom-in on Brussels.

Note that the given computational times also include the time required to read the instance and to pre-process distance information. If one neglects this setup time, it does not take more than 3 s to compute an initial solution for the largest instance F2. Given a few minutes up to a few hours of computational time, the initial solutions can be improved by more than 6% with local search. It appears that KGLS^{XXL} can improve the solutions of instances with longer routes and an eccentric depot location more drastically over

Table 6
Results on the new very large-scale instances for KGLS^{XXL} and LKH-3 for different computational times T in minutes. The gap is computed with respect to KGLS^{XXL}. The depot location D can be central (C) or eccentric (E), and the average route length (L) varies across instances.

Instance (N)	D	L	CW			KGLS^{XXL} (short time)			KGLS^{XXL}		LKH-3		
			Value	Gap	T	Value	Gap	T	Value	T	Value	Gap	T*
L1 (3,000)	C	14.8	200,957	3.34	0.1	194,528	0.04	15.0	194,456	60.0	194,381	−0.04	> 3000.0
L2 (4,000)	E	85.1	126,122	9.85	0.1	116,479	1.45	20.0	114,817	80.0	113,484	−1.16	> 3000.0
A1 (6,000)	C	17.5	497,441	3.29	0.2	482,078	0.10	30.0	481,583	120.0	481,338	−0.05	> 3000.0
A2 (7,000)	E	58.3	322,073	8.79	0.2	300,277	1.43	35.0	296,055	140.0	297,478	0.48	> 3000.0
G1 (10,000)	C	20.6	489,556	3.38	0.4	474,382	0.17	50.0	473,568	200.0	474,164	0.13	> 3000.0
G2 (11,000)	E	100	288,942	9.24	0.4	267,354	1.07	55.0	264,512	220.0	265,763	0.47	> 3000.0
B1 (15,000)	C	29.3	531,980	4.91	0.9	510,218	0.61	75.0	507,103	300.0	509,457	0.46	> 3000.0
B2 (16,000)	E	87.9	384,437	8.06	1.1	361,804	1.69	80.0	355,779	320.0	357,382	0.45	> 3000.0
F1 (20,000)	C	29.2	7,518,845	3.06	1.7	7,326,975	0.43	100.0	7,295,447	400.0	7,300,772	0.07	> 3000.0
F2 (30,000)	E	117.2	4,803,502	6.64	3.8	4,566,287	1.37	150.0	4,504,416	600.0	4,499,422	−0.11	> 3000.0
Average				6.05	0.6		0.84	61.0		244.0		0.07	> 3000.0

*Results were obtained after several days up to a month of computational time, the exact times could not be retrieved (Helsgaun, 2018).

time, which might be due to the fact that it is more difficult to compute good initial solutions.

We further obtained results from Helsgaun (2017) (LKH-3). This heuristic is an extension of a heuristic for the TSP (Helsgaun, 2000) and transforms a VRP into a standard TSP, while checking for constraint violations after each λ-opt move. Whereas a move is performed in $O(\sqrt{n})$ time, a constraint violation check takes $O(n)$ time. The heuristic focuses on solution quality rather than speed and, thus, the results were obtained after several days up to a month of computational time. LKH-3 produces results comparable to those of KGLS^{XXL} for the new instances, which suggests that heuristics that work well for TSPs can also be adapted to effectively solve VRPs, especially if variation in demand is low (between 1 and 3) and routes visit many customers. LKH-3 also computed excellent solutions for the $\mathbb{W}$-instances with very long routes which were more than 2% better than the ones reported above. We want to remark that it is difficult to make a statement about the absolute quality of the reported solutions, since no decent lower bounds could be derived to estimate the optimality gap.

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 reasonable amount of computational time. We outlined the main challenges of very large-scale problems — reduction of time and space complexity — and demonstrated how to tackle them in the context of the VRP by limiting information storage and the size of neighborhoods with heuristic pruning. These ideas were embedded into an already 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, both in the construction of initial solutions and in the improvement with local search. The resulting heuristic significantly improves the results of a previous heuristic in relatively short computational times. 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., Rohe, A., 2003. Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. Comput. 15 (1), 82–92.

Arnold, F., Sörensen, K., 2018. What makes a VRP solution good? The generation of problem-specific knowledge for heuristics. Comput. Oper. Res. doi:10.1016/j.cor.2018.02.007. Advance online publication

Arnold, F., Sörensen, K., 2019. Knowledge-guided local search for the vehicle routing problem. Comput. Oper. Res. 105, 32–46.

Assad, A.A., Golden, B.L., 1995. Arc Routing Methods and Applications. In: Handbooks in Operations Research and Management Science, vol. 8, pp. 375–483.

Buhrkal, K., Larsen, A., Ropke, S., 2012. The waste collection vehicle routing problem with time windows in a city logistics context. Procedia-Soc. Behav. Sci. 39, 241–254.

Cattaruzza, D., Absi, N., Feillet, D., González-Feliu, J., 2017. Vehicle routing problems for city logistics. EURO J. Transp. Logist. 6 (1), 51–79.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. 12 (4), 568–581.

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

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

Helsgaun, K., 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. Eur. J. Oper. Res. 126 (1), 106–130.

Helsgaun, K., 2017. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems: Technical Report.

Helsgaun, K., 2018. Results of LKH-3. Personal communication. 2018-02-20

Irnich, S., Funke, B., Grünert, T., 2006. Sequential search and its application to vehicle-routing problems. Comput. Oper. Res. 33 (8), 2405–2429.

Janssens, J., Van den Bergh, J., Sörensen, K., Cattrysse, D., 2015. Multi-objective microzone-based vehicle routing for courier companies: From tactical to operational planning. Eur. J. Oper. Res. 242 (1), 222–231.

Johnson, D.S., Papadimitriou, C.H., Yannakakis, M., 1988. How easy is local search? J. Comput. Syst. Sci. 37 (1), 79–100.

Kytöjoki, J., Nuortio, T., Bräysy, O., Gendreau, M., 2007. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. Comput. Oper. Res. 34 (9), 2743–2757.

Laporte, G., 2007. What you should know about the vehicle routing problem. Nav. Res. Logist. (NRL) 54 (8), 811–819.

Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. 21 (2), 498–516.

Mester, D., Bräysy, O., 2007. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Comput. Oper. Res. 34 (10), 2964–2975.

PassMark Software, 2018. Cpu benchmarks. https://www.cpubenchmark.net/. Accessed: 2018-02-05.

Patier, D., Routhier, J.-L., 2009. How to improve the capture of urban goods movement data? In: Transport Survey Methods: Keeping up with a Changing World. Emerald Group Publishing Limited, pp. 251–287.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017. Improved branch-cut-and-price for capacitated vehicle routing. Math. Program. Comput. 9 (1), 61–100.

Reinelt, G., 1991. TSPLIB–A traveling salesman problem library. ORSA J. Comput. 3 (4), 376–384.

Subramanian, A., Uchoa, E., Ochi, L.S., 2013. A hybrid algorithm for a class of vehicle routing problems. Comput. Oper. Res. 40 (10), 2519–2531.

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

Toth, P., Vigo, D., 2003. The granular tabu search and its application to the vehicle routing problem. INFORMS J. Comput. 15 (4), 333–346.

Toth, P., Vigo, D., 2014. Vehicle Routing: Problems, Methods, and Applications. SIAM.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A., 2017. New benchmark instances for the capacitated vehicle routing problem. Eur. J. Oper. Res. 257 (3), 845–858.

Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W., 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. 60 (3), 611–624.

Voudouris, C., Tsang, E.P., 2003. Guided Local Search. Springer.

Wøhlk, S., Laporte, G., 2018. A fast heuristic for large-scale capacitated arc routing problems. J. Oper. Res. Soc. 127–141.

Zhong, H., Hall, R.W., Dessouky, M., 2007. Territory planning and vehicle dispatching with driver learning. Transp. Sci. 41 (1), 74–89.