Volume 37 (1), pp. 17–53
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)

Vehicle routing decision support for a local retailer

AD Shuttleworth* JH van Vuuren†

Received: 8 January 2021; Revised: 31 May 2021; Accepted: 31 May 2021

Abstract

One of the most challenging decisions that has to be made routinely by dispatch managers at distribution centres of warehousing and distribution businesses in the retail sector involves the assignment of delivery vehicles to service customers exhibiting demand for retail goods and the subsequent routing of these delivery vehicles to the various customers and back again. Perhaps surprisingly, these dispatch managers do not always use vehicle routing software to schedule goods deliveries to customers, instead often relying on teams of human schedulers who perform this task manually. The reason for not using such software is usually a perception that it may be difficult to integrate the software with existing enterprise resource planning systems already in use. In such cases, estimates of potential costs savings that may be brought about by appropriate software is often required before the dispatch department will risk the significant step towards investing in vehicle routing planning software. Dispatch managers may then employ these cost savings estimates in cost-benefit trade-off analyses. This paper contains a practical case study in which the potential cost savings of a vehicle routing optimisation approach are quantified for a large retail distribution centre in the South African Western Cape in a bid to support its decision as to whether or not to invest in vehicle routing planning software.

Key words: Vehicle routing, decision support, case study.

1 Introduction

It is well known that *distribution centres* (DCs) form a crucial part of a retail supply chain, facilitating the shipment of retail goods from suppliers to end customers. Products are usually received from different suppliers at a DC which acts as a central hub at which all of these products are consolidated, sorted, picked and dispatched in appropriate quantities to customers [22]. This type of distribution hub makes sense from both the perspectives

*Stellenbosch Unit for Operations Research in Engineering, Department of Industrial Engineering, Stellenbosch University, Private Bag X1, Matieland, 7602, South Africa, email: aaronshuttleworth@gmail.com

†Corresponding author: **(Fellow of the Operations Research Society of South Africa)** Stellenbosch Unit for Operations Research in Engineering, Department of Industrial Engineering, Stellenbosch University, Private Bag X1, Matieland, 7602, South Africa, email: vuuren@sun.ac.za

http://dx.doi.org/10.5784/37-1-691

of reducing the lead time associated with customer demand fulfilment and leveraging economies of scale by the retailer in question.

The management of operations at a retail DC is typically complex, requiring challenging decisions in respect of DC layout, inventory replenishment, and customer order picking, to name but a few. But perhaps one of the most challenging decisions that routinely has to be made by the dispatch departments of warehousing and distribution businesses in the retail sector involves the assignment of delivery vehicles to service customers exhibiting demand for retail goods and the subsequent routing of these delivery vehicles from the DC to the various customers and back to the DC again. The SPAR Group Ltd is one such a warehousing and distribution business. It is listed on the *Johannesburg Securities Exchange* (JSE) [31]. Headquartered in Durban, South Africa, SPAR Group Ltd operates in nine different countries, has ten DCs and serves 3 768 SPAR retail stores in total.

One of its DCs is located in Phillipi, in the Western Cape. Servicing 284 South African SPAR stores in January 2020 alone, the SPAR Western Cape DC scheduled between 174 and 303 individual deliveries on any given day of the month, save for Saturdays [1]. Deliveries are usually scheduled the day before, and no deliveries are made on Sundays. The stores serviced by the SPAR Western Cape DC are situated in the Western Cape and portions of the Northern Cape. The DC has access to an active fleet of more than 40 vehicles of varying sizes, as may be seen in Figure 1, which services stores within a travel radius of approximately 150km from the DC [10]. Additional third-party vehicles may, however, also be hired and are typically used to service more distant stores.

Figure 1: A selection of delivery vehicles of different sizes parked at the outbound loading bays of the SPAR Western Cape DC.

Delivery vehicle dispatch decisions at retail DCs typically involve solving instances of the celebrated *vehicle routing problem* (VRP). The VRP is a combinatorial optimisation problem in which a set routes is sought for a fleet of delivery vehicles that collectively services a number of customers[1] and which minimises the total distance travelled by all the vehicles. The archetypal VRP prototype is the *capacitated VRP with time windows* (CVRPTW) in which each vehicle additionally has a specified capacity (upper bound) in terms of the volume of goods that it can transport to customers and in which visits by vehicles assigned to service customers have to be timed so that these customers are serviced within pre-specified time windows. Large instances of the CVRPTW are notoriously difficult to solve exactly and the highly combinatorial nature of these problem instances makes it extremely difficult to construct high-quality approximate solutions manually. A variety of specialised commercial software options therefore exist for supporting delivery vehicle dispatch decisions at DCs [9, 23, 24, 28, 29].

[1]Each customer has to be visited by exactly one vehicle for the purpose of goods delivery.

The SPAR Western Cape DC does not, however, use vehicle routing software to schedule its deliveries. Instead, a team of human schedulers performs this task manually. While this may come as a surprise to the reader, it is our experience that this seemingly precarious situation is often the case in the South African retail sector. Reasons for not using specialised software to streamline vehicle routing dispatch decisions may be related to the often prohibitive cost of such software or may simply be the result of such software being difficult to integrate with existing *enterprise resource planning* (ERP) systems already in use. It is often difficult for a retailer to take the significant step towards investing in VRP planning software for the aforementioned reasons, among others.

We were therefore tasked by the SPAR Western Cape DC to assist its management in determining whether such an investment step would be appropriate in terms of the expected cost savings that may be brought about by VRP planning software. We did this by quantifying the range of potential savings in the cost associated with its delivery vehicle routing decisions that may be realised if a VRP optimisation approach were to be adopted when performing these decisions as opposed to performing these decisions manually based on (considerable) historical route scheduling experience. We anticipate that the experience that we gained during this process may be useful for operations research consultants and so we decided to write up this cost saving quantification process as a practical case study.

The remainder of this paper is organised as follows. Upon having conducted a brief review in §2 of the literature with the aim of identifying powerful CVRPTW solution approaches capable of solving realistically sized CVRPTW instances approximately, we derive a mixed-binary programming CVRPTW model in §3 that we subsequently set about solving in the context of the SPAR Western Cape DC dispatch decisions with a view to quantify the aforementioned potential cost savings. Thereafter, we show that the computational complexity of this model is such that its exact solution seems to be out of reach of a state-of-the-art mixed-integer programming solver and standard personal computing technology. Based on our findings during an exploratory literature review, we describe an acclaimed hybrid metaheuristic in §4 selected for the purpose of solving approximately three representative CVRPTW model instances in a special case study in §5, involving dispatch data obtained from the SPAR Western Cape DC. In doing so, we demonstrate that the hybrid metaheuristic is capable of achieving considerable dispatch cost savings. The paper finally closes with a number of concluding remarks in §6.

2 Literature review

In their most basic form, exact methods for solving VRP instances are based on the branch-and-bound method, the branch-and-cut method and the branch-and-price method. The last half-century has, however, seen a monumental volume of research on the design and improvement of methods for solving VRP instances exactly. While efforts continue to develop more powerful exact mathematical programming techniques, usually involving the implementation of ever-more specialised and sophisticated types of cuts, Toth and Vigo [35] have claimed that these exact methods can only reasonably be applied to VRP instances of approximately one hundred customers or fewer. For larger problem instances, the use of (meta)heuristics becomes necessary in order to obtain high- quality solutions within a reasonable time.

Heuristics have been a part of VRP research for as long as the VRP itself has been researched explicitly. In their landmark 1959 paper, Dantzig and Ramser [8] described the VRP for the first time, as well as a simple heuristic for solving VRP instances that seeks to eliminate partial solutions through trial and error. Just as newer and better exact VRP algorithms have been proposed over the years, so VRP heuristics have also made significant advances. One of the most popular VRP heuristics, however, remains the savings method proposed in 1964 by Clark and Wright [4].

Simple heuristics are, however, constrained in how useful they are on their own. In 1986, Glover [11] coined the term *metaheuristic* to describe a category of heuristic search methods that pursue alternative solutions even after having encountered local optima. These methods are often considered to be heuristics guiding other heuristics. Since the dawn of the new millennium, metaheuristics that have appeared in the literature generally reside in one of two major classes: Trajectory-based metaheuristics and population-based metaheuristics. The former iteratively seek to improve a single solution by exploring solutions locally, while the latter maintain a population of simultaneous solutions that are recombined iteratively to create new populations of solutions [40].

Recently, hybrid metaheuristics have risen to prominence in the VRP literature. These methods seek to combine the strengths of both trajectory-based and population-based approaches in order to provide more powerful and more flexible algorithms. Hybrid metaheuristics can be applied to a large variety of VRP variations without the need for implementing major algorithmic changes in order to accommodate particular problem attributes. Their strengths include allowing for VRP instances to be solved in consistently shorter times and returning approximate solutions that are less than a percent away from the best solutions documented for well-known benchmark problems [35].

Vidal *et al.* [40] identified the most common metaheuristics applied to VRP instances exhibiting a variety of attributes in an important survey paper of 2013. They documented the frequencies of use of these metaheuristics, as summarised in Table 1. Hybridisations of these metaheuristics have also been applied frequently (thirty nine of the sixty five algorithms surveyed, in fact, featured some form of hybridisation).

Trajectory-based	**Freq**	**Population-based**	**Freq**
Tabu search	17	Genetic or evolutionary algorithm	16
Iterated local search	7	Ant colony optimisation	4
Variable neighbourhood search	5	Scatter search	2
Adaptive large neighbourhood search	4	Path relinking	2
Simulated annealing	3	Particle swarm optimisation	1
Others	4		

Table 1: Frequency of metaheuristics used in best-performing VRP solution methods according to the survey by Vidal *et al.* [40].

Both Toth and Vigo [35] and Vidal *et al.* [40] presented computational results obtained by a variety of metaheuristics in the context of the two most commonly adopted VRP benchmark data sets. The algorithmic performance data reviewed by Toth and Vigo [21] are recounted here, as much of these data appear to be the same as those considered by Vidal *et al.* [40]. In particular, the performance data presented were obtained by the

metaheuristics listed in Table 2 in the context of a data set proposed earlier by Golden *et al.* [12] and a data set proposed by Christofides *et al.* [2]. The former data set contains the larger instances of the two, with twenty instances each containing between 240 and 483 customers, while the latter data set consists of fourteen instances, each containing between 50 and 200 customers. Since the number of customers encountered later in the case study of this paper ranges from 200 to 300 customer stops, considering the larger set of instances would seem more appropriate in the context of this paper.

Acronym	**Reference**	**Algorithmic approach**
CLM01	Cordeau *et al.* [5]	Tabu search
TV03	Toth and Vigo [34]	Granular tabu search
RDH04	Reimann *et al.* [27]	Ant colony optimisation
T05	Tarantilis [33]	Adaptive memory + tabu search
MB07	Mester and Bräysy [19]	Evolutionary algorithm + local search
PR07	Pisinger and Ropke [25]	Adaptive large neighbourhood search
NB09	Nagata and Bräysy [21]	Hybrid genetic algorithm
P09	Prins [26]	Greedy randomised adaptive search procedure + evolutionary local search
GGW10	Groër *et al.* [13]	Route-to-route + evolutionary computation
ZK10	Zachariadis and Kiranoudis [41]	Guided local search + tabu search
GGW11	Groër *et al.* [14]	Parallel route-to-route
CM12	Cordeau and Maischberger [6]	Parallel iterated tabu search
JCL12	Jin *et al.* [15]	Parallel cooperative tabu search
VCGLR12	Vidal *et al.* [38]	Hybrid genetic algorithm
SUO13	Subramanian *et al.* [32]	Set partitioning + iterated local search

Table 2: Top-performing VRP metaheuristics compared by Toth and Vigo [35]. Adapted from Toth and Vigo [35] and from Vidal *et al.* [40].

Table 3 contains a summary of the average percentage gap achieved between each algorithm's best solution and the overall best solution known for the benchmark instances, along with the actual run time and normalised[2] run time in each case. These average gap percentages and normalised run times are presented graphically in Figure 2.

According to the results in Figure 2, the algorithms achieving the most impressive trade-offs between computation time expended and solution quality achieved (P09, VCGLR12-A, VCGLR12- B and MB07) are all hybrid metaheuristics. A plausible explanation for the success of hybrid metaheuristics is that they combine the strength of population-based algorithms in terms of a thorough exploration of the search space with the strength of trajectory-based algorithms in terms of exploitation of local optima.

In 2013, Vidal *et al.* [39] proposed the *Hybrid Genetic Search with Advanced/Adaptive Diversity Control* (HGSADC) algorithm for solving CVRPTW instances. This algorithm is similar to the one proposed the year before by Vidal *et al.* [38], but features the addition of decomposition phases to the HGSADC algorithm which are beneficial when solving large problem instances. Results obtained by the original version of the algorithm (the square data points VCGLR12-A and VCGLR12-B) appear on the Pareto frontier of the

[2]The run times were normalised in order to account for differing CPU processing capabilities and thus to allow for a fairer comparison between metaheuristics implemented sequentially and in parallel, although this does incur a slight penalty for metaheuristics implemented in parallel due to their communication overheads. (The normalised times are only estimates based on processor power and sequential run time.)

(Meta)heuristic	Configuration	Optimality gap (%)	Normalised time (s)	Clock time (s)
GGW11-A	129 threads, best of 5 runs	0.12	138 899	193 500
VCGLR12-A	Average of 10 runs, 50 000 iterations	0.16	3 387	5 563
NB09-A	Best of 10 runs	0.17	13 006	21 359
ZK10-A	Best of 10 runs	0.22	14 128	24 300
VCGLR12-B	Average of 10 runs, 10 000 iterations	0.27	1 042	1 712
NB09-B	Average of 10 runs	0.27	1 301	2 136
GGW11-B	8 threads, best of 5 runs	0.30	8 470	11 800
MB07-A	Best configuration	0.33	782	1 461
JCL12-A	8 threads, best of 10 runs	0.35	200 978	200 978
SUO13-A	Best of 10 runs	0.40	39 382	39 382
ZK10-B	Average of 10 runs	0.43	1 413	2 430
GGW11	4 threads, best of 5 runs	0.44	4 235	5 900
CM12-A	Best of 10 runs, 1 000 000 iterations	0.56	18 770	18 770
SUO13-B	Average of 10 runs	0.56	3 938	3 938
JCL12-B	8 threats, average of 10 runs	0.60	20 098	20 098
P09	—	0.63	233	436
PR07-A	Best of 10 runs, 50 000 iterations	0.82	3 662	6 457
T05	Standard	0.93	169	2 729
RDH04	—	0.93	599	2 960
CM12-B	Average of 10 runs, 1 000 000 iterations	0.94	1 877	1 877
GGW10-A	Ejection – Random	1.19	43	55
MB07-B	Fast configuration	1.23	7	13
GGW10-B	Set partitioning	1.27	10	13
PR07-B	Average of 10 runs, 50 000 iterations	1.35	366	646
CM12-C	Average of 10 runs, 500 000 iterations	1.46	188	188
CLM01	—	1.79	1 207	3 366.6
TV03	—	3.21	21	1 053

Table 3: Computational results achieved by the metaheuristics listed in Table 2 when solving the VRP benchmark instances of Golden *et al.* [12]. The percentage gap from the best-known solution, the normalised run time, and the clock time are shown. Adapted from Toth and Vigo [35].

benchmark results in Figure 2 (the data points in red). The HGSADC algorithm achieved the highest-quality solutions within a reasonable time (of respectively no more than 75 minutes and 17 minutes, on average, per instance). These data made the 2013 algorithmic variation our algorithm of choice when solving the (relatively large) CVRPTW instances considered in the case study of this paper.

3 Mathematical model

In the spirit of reproducibility and for the sake of future comparisons of results, this section is devoted to a detailed discussion on the CVRPTW model considered in this paper. The model is derived in the form of a mixed-binary programming problem in §3.1 and its implementation in a state-of-the-art mixed-integer solver and subsequent verification are described in §3.2. The complexity of the implementation is finally measured in §3.3.

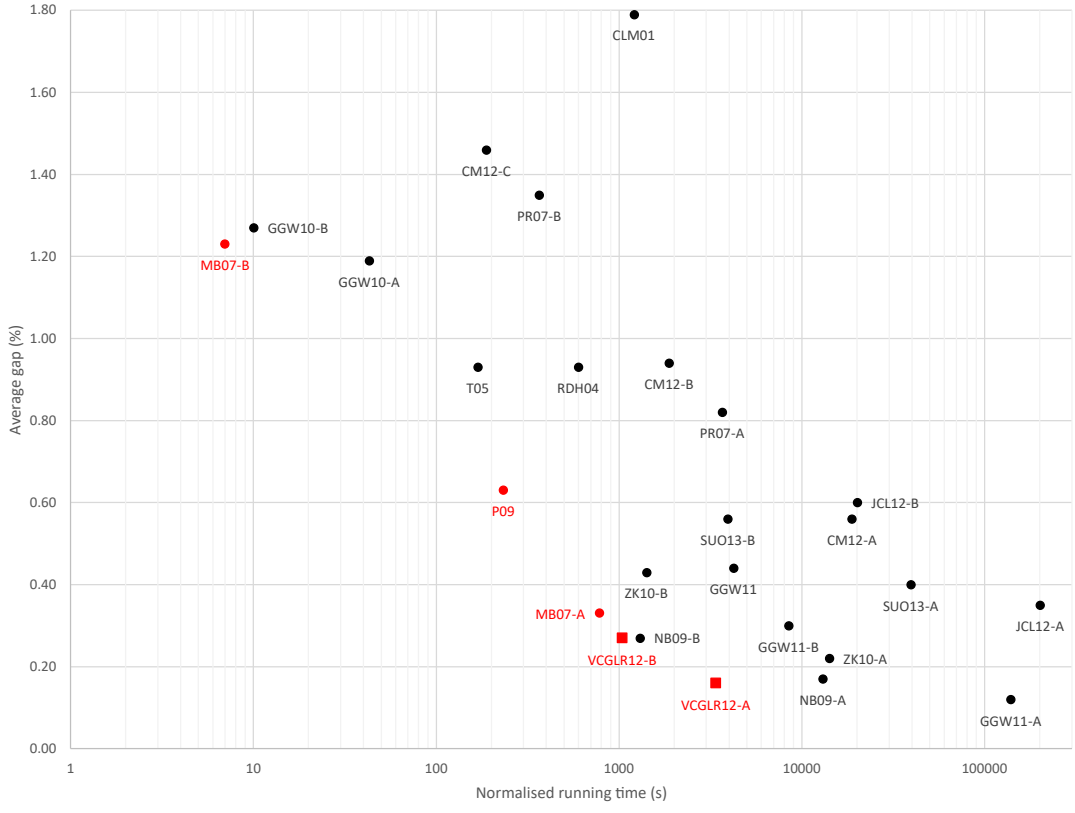

Figure 2: Computational results reported for the metaheuristics listed in Table 3 when solving the benchmark instances of Golden *et al.* [12], displayed in a log-linear scatter graph. The percentage gaps from best-known solutions and normalised run times (measured in seconds) are displayed. Adapted from Toth and Vigo [35].

3.1 Model derivation

The CVRPTW model considered in this paper is derived in this section. The various input parameters required for an instance of the model are described in §3.1.1. The model variables are next declared in §3.1.2, while §3.1.3 is devoted to a detailed derivation of and motivation for the various model constraints. The model objective function is finally derived in §3.1.4.

3.1.1 Model parameters

Let the set $\mathcal{L} = \{0, 1, \ldots, m, m+1\}$ index m customers who have to be serviced from a central depot. As is customary in the VRP literature, the depot itself is represented by the indices 0 and $m+1$ for vehicle departure and return, respectively. "Customers" 0 and $m+1$ therefore share the same location. Also, denote the number of pallets of demand experienced by customer $j \in \mathcal{L} \setminus \{0, m+1\}$ by Q_j, with the demand of the depot being zero.

Denote the start and end times of the time window during which a vehicle may service

customer $j \in \mathcal{L}$ by W_j^S and W_j^E, respectively, both measured in hours after midnight. If any customer j does not specify such a time window, the parameter values $W_j^S = 0$ and $W_j^E \geq 24$ are assumed (this is the case for the depot as well, as it is always open). Denote the average unload time per pallet at customer $j \in \mathcal{L} \setminus \{m+1\}$ by U_j, measured in hours, and the expected time required to load a pallet at the depot by U_0.

Moreover, let the set $\mathcal{V} = \{1, 2, \ldots, n\}$ index the vehicles available to service customers, and denote the cost per kilometre and the cost per hour expended by vehicle $k \in \mathcal{V}$ by C_k^D and C_k^T, respectively. Suppose the pallet loading capacity of vehicle $k \in \mathcal{V}$ is C_k^P, and that the expected travel distance and travel time from customer $i \in \mathcal{L}$ to customer $j \in \mathcal{L}$ are denoted by d_{ij} and t_{ij}, respectively.

3.1.2 Model variables

Define the binary decision variable

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \in \mathcal{V} \text{ should travel from customer } i \in \mathcal{L} \text{ to customer } j \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

and capture the number of pallets delivered to customer $j \in \mathcal{L} \setminus \{m+1\}$ (or being loaded at the depot in the case of $j = 0$) by vehicle $k \in \mathcal{V}$ in an integer decision variable p_{jk}. This variable allows for split deliveries, as the demand experienced by some customers may not be serviceable by a single vehicle.

While it can, in fact, be derived from the decision variables and model parameters, an auxiliary variable s_{jk} is defined in order to simplify the model constraints, capturing the expected time (measured in hours) at which vehicle $k \in \mathcal{V}$ will begin its service at customer $j \in \mathcal{L}$. A vehicle may arrive at a customer before the service window of that customer starts, but can only start to unload (commence service) within the window. Note that s_{0k} represents the time at which vehicle $k \in \mathcal{V}$ starts loading at the depot and that $s_{m+1,k}$ denotes the time at which the vehicle returns to the depot. A second auxiliary variable z_{ijk} is required to enable the objective function to be written in a linear form. This non-negative variable either represents the difference between the times when vehicle $k \in \mathcal{V}$ begins service at customers $i \in \mathcal{L}$ and $j \in \mathcal{L}$ if vehicle k indeed travels from customer i to customer j, or is considered meaningless otherwise.

3.1.3 Model constraints

A number of constraints are imposed in the model so as to ensure the practical feasibility of solutions. In order to ensure that the demand of customer $j \in \mathcal{L} \setminus \{0, m+1\}$ is met, the constraint set

$$\sum_{k \in \mathcal{V}} p_{jk} = Q_j, \quad j \in \mathcal{L} \setminus \{0, m+1\} \tag{1}$$

is enforced. Moreover, to allow later constraints to accommodate the time it takes to load each vehicle at the depot, the number of pallets loaded into vehicle k at the depot is required to match the sum total of all pallets being delivered by the vehicle. The constraint

set

$$p_{0k} = \sum_{j \in \mathcal{L} \setminus \{0, m+1\}} p_{jk}, \quad k \in \mathcal{V} \tag{2}$$

enforces this. A vehicle, of course, cannot service a customer if it does not visit this customer. In the constraint set

$$p_{jk} \leq C_k^P \sum_{i \in \mathcal{L}} x_{ijk}, \quad j \in \mathcal{L} \setminus \{0, m+1\},\ k \in \mathcal{V}, \tag{3}$$

the right-hand side equals zero if vehicle k is not assigned to service customer j. The constraint set therefore prevents the vehicle from servicing that customer. Each vehicle should also depart from the depot exactly once if it is assigned to service customers at all, and it may not depart with a pallet load greater than the vehicle's capacity. The constraint set

$$p_{0k} \leq C_k^P \sum_{j \in \mathcal{L} \setminus \{0, m+1\}} x_{0jk}, \quad k \in \mathcal{V} \tag{4}$$

ensures this. The depot itself should be excluded from the set of customers in this case. Similarly, each vehicle is required to return to the depot exactly once after having departed from the depot. Applying the constraint set

$$\sum_{j \in \mathcal{L} \setminus \{0, m+1\}} x_{0jk} - \sum_{i \in \mathcal{L} \setminus \{0, m+1\}} x_{i,m+1,k} = 0, \quad k \in \mathcal{V} \tag{5}$$

enforces this, while the constraint set

$$\sum_{j \in \mathcal{L} \setminus \{0, m+1\}} x_{0jk} \leq 1, \quad k \in \mathcal{V} \tag{6}$$

allows vehicles to depart from the depot only once each. The constraint sets

$$\sum_{i \in \mathcal{L}} x_{i0k} = 0, \quad k \in \mathcal{V} \tag{7}$$

and

$$\sum_{j \in \mathcal{L}} x_{m+1,jk} = 0, \quad k \in \mathcal{V} \tag{8}$$

are furthermore included in a bid to reduce the model solution time. These constraints are not necessary to find feasible solutions, as x_{i0k} and $x_{m+1,jk}$ are meaningless in the final output. Informal empirical experimentation, however, showed that the inclusion of constraints (7)–(8) leads to a significant decrease in the time required to solve the model.

By ensuring that the difference between the sum total of all arrivals at customer j and the sum total of all departures from customer j remains zero for each vehicle k, the constraint set

$$\sum_{i \in \mathcal{L} \setminus \{j\}} x_{ijk} - \sum_{i \in \mathcal{L}} x_{jik} = 0, \quad j \in \mathcal{L} \setminus \{0, m+1\},\ k \in \mathcal{V} \tag{9}$$

limits each vehicle to depart from a given customer the same number of times that it arrives there (either once or not at all). Constraint set (9) also prevents vehicles from

"departing" and subsequently "arriving" at the same customer (thus avoiding loops), as the exclusion of customer j from the first sum means that the constraint set is not satisfied if $x_{jjk} = 1$.

With constraints in place ensuring that demand is met and that vehicles are assigned routes that begin and end at the depot, the following constraints ensure solution feasibility with respect to the time windows. The auxiliary variable s_{jk}, representing the time at which vehicle k starts servicing customer j, has to be constrained to within the given customer's time window. The constraint set

$$W_j^S \leq s_{jk} \leq W_j^E, \quad j \in \mathcal{L},\ k \in \mathcal{V} \tag{10}$$

ensures that if a vehicle arrives at a customer before the start of that customer's time window, it can only start unloading at (*i.e.,* servicing) the customer once its service time window begins. Each side of the constraint set (10) could have been multiplied by $\sum_{i \in \mathcal{L}} x_{ijk}$ to render $s_{jk} = 0$ if vehicle k does not visit customer j. This would, however, have introduced non-linearity into the model. The value of the variable s_{jk} can instead simply be considered meaningless if vehicle k does not visit customer j.

It should further be ensured that every vehicle arrives at each customer to which it has been assigned service before the end of the customer's time window. While constraint set (10) ensures that the service only starts within the customer's time window, the constraint set

$$s_{ik} + p_{ik}U_i + t_{ij} \leq s_{jk} + y_{ij}(1 - x_{ijk}), \quad i, j \in \mathcal{L},\ k \in \mathcal{V} \tag{11}$$

ensures that the expected time at which service starts at customer i, plus the expected unloading time at customer i, plus the expected travel time from customer i to customer j is no more than the time at which service at customer j starts. The right-hand side of the above constraint set ensures that the constraint is only enforced if vehicle k travels from customer i to customer j. This requires the constant $y_{ij} = \max\{W_i^E + Q_i U_i + t_{ij} - W_j^S, 0\}$ for each customer pair $i, j \in \mathcal{L}$.

The constraint set

$$z_{ijk} \geq s_{jk} - s_{ik} - M(1 - x_{ijk}), \quad i, j \in \mathcal{L},\ k \in \mathcal{V} \tag{12}$$

is required to allow the objective function to be represented linearly. In this constraint set, the value z_{ijk} must be at least the difference between between s_{jk} and s_{ik} (the expected times at which vehicle k begins service at customers j and i, respectively) if the vehicle travels from customer i to customer j (*i.e.,* if $x_{ijk} = 1$). Recall that z_{ijk} is non-negative. In (12), $M = \max\{W_j^E\} - \min\{W_j^S\}$, where the maximum and minimum are taken over all customers $j \in \mathcal{L}$. The fact that z_{ijk} will be included in the objective function, which has to be minimised, means that the variable itself will be minimised to zero if the value of x_{ijk} is zero.

It is implicitly assumed that multiple vehicles can unload simultaneously at the same customer. For all time-based parameters and variables, it is possible to consider a period longer than a day by extending W_j^S and W_j^E for all appropriate customers to the desired number of hours after the reference time 0. There may, however, only be one time window per customer for the entire period under consideration.

3.1.4 Objective function

The objective pursued in the model is to

$$\text{minimise} \sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{L}}\sum_{k\in\mathcal{V}}\left(C_k^D d_{ij}x_{ijk} + C_k^T z_{ijk}\right). \tag{13}$$

This cost function takes into account the distance over which each vehicle travels and the time that it spends waiting and unloading at customers, as well as the time it spends travelling between customers.

If vehicle k does not travel from customer i to customer j (*i.e.,* if $x_{ijk} = 0$), there is no contribution the objective function associated with that vehicle. If, however, vehicle k does indeed travel from customer i to customer j (*i.e.,* if $x_{ijk} = 1$ and $z_{ijk} > 0$), the expected cost of both distance and time are taken into consideration. The cost of travelling is accounted for by multiplying C_k^D (the cost per kilometre of vehicle k) by d_{ij} (the expected travel distance from customer i to customer j). The cost of time expenditure is accounted for by multiplying C_k^T (the cost per hour of vehicle k) by z_{ijk} (the expected time difference between commencing service at customer j and commencing service at customer i).

3.2 Model implementation and verification

The model derived in §3.1 was implemented in IBM ILOG's CPLEX Optimisation Studio 12.10 (CPLEX), which is a state-of-the-art software suite for solving mixed-integer and linear
programming problems. After constructing an initial solution by root node processing, CPLEX applies the branch-and-cut method to find an optimal solution to a mixed integer programming problem instance.

We subsequently verified our model implementation by applying the model to eighty small, randomly generated hypothetical test problem instances. These test instances were generated by sampling methodically randomised input parameters. Any set of input data for which CPLEX found no feasible solution was discarded upon which the problem instance was randomised again. The following method was utilised during this randomisation process:

1. Three vehicle types were conceived and assigned the pallet capacities 16, 22, and 30, respectively.
2. Each vehicle type was assigned a random distance cost (a real number between 0.7 and 1.5) a random time cost (a real number between 7 and 15).
3. A predetermined number of vehicles were created, each assigned properties (cost-per-km, cost-per-hour, and pallet load capacity) matching a random vehicle type.
4. "Customers" 0 and $m + 1$ were both assigned coordinates $(0, 0)$ in the Cartesian plane.
5. Each customer was assigned random coordinates (real numbers between -100 and 100) in the Cartesian plane.

6. The distances between locations were calculated as the Euclidean distances between the coordinates of customers in the Cartesian plane.
7. The travel times between customers were calculated as the distance divided by 80.
8. The sum of all vehicle capacities was divided by the number of customers. Each customer was assigned a random integer demand between 1 and this number.
9. With a probability $N_{\text{vehicles}}/(2N_{\text{customers}})$, customers were assigned 1-hour delivery windows, the starts of which were integers between 5 and 23. Here and in the remainder of the paper N_x denotes "the number of x." If, however, customer j was not assigned a delivery time window, its window start and end times, W_j^S and W_j^E, were set to 0 and 24, respectively.
10. Every customer was assigned a random per-pallet unload time between 0.01667 and 0.1667, corresponding to 1 and 10 minutes, respectively.

For each test problem instance generated as described above, the model solution output data were inspected and compared with the input parameters so as to verify solution feasibility. In particular, the output data aspects that were inspected for validity included the following:

- Each vehicle had to be assigned a circular route, departing first from the depot ("customer" 0), visiting every customer that it had to service exactly once along its route, and finally had to return to the depot ("customer" $m + 1$) again.
- The sum of all delivery quantities assigned to a vehicle had to be no more than its capacity.
- The sum of all delivery quantities received by a customer had to be equal to the demand of that customer.
- The service start time at any given customer had to be within the delivery window of that customer.
- The service start time of any customer had to be at least as large as the service start times of all preceding customers visited along the vehicle's route.

Upon inspection, it was found that every solution returned by the model implemented in CPLEX satisfied all of the above criteria, and so the model was considered verified in the *program testing with test data* sense, as recommended by Kendall and Kendall [17]. Many of the parameter values selected for the above verification process were arbitrary, serving merely as a means to verify that the solutions returned by the model implementation indeed satisfy all the model constraints. For example, the probability of customers having delivery time windows associated with them could have been raised, but this would increase the portion of random problem instances that have no feasible solutions.

Some of the numbers mentioned above were, however, not chosen completely arbitrarily. For example, the impact of time and distance on the objective function was balanced, with roughly the same order of magnitude for the final values. Moreover, the range of possible vehicle capacity values was based on SPAR Western Cape DC's actual largest and smallest vehicle capacities. The DC's customers who actually have negotiated time windows associated with them also all have one-hour time windows. Finally, stores' per-pallet unload times often vary between one and ten minutes.

3.3 Model time complexity

In order to quantify the correlation between the problem instance dimensions and the required corresponding solution time, batches of test problem instances were generated randomly and solved. Each batch consisted of ten randomised problem instances, and the solution times were recorded. The numbers of customers and vehicles varied between batches, but were kept fixed within each batch. In particular, the number of customers varied from six to fifteen, while the number of vehicles was taken as three or five. The mean solution times associated with these batches are illustrated graphically in Figure 3.

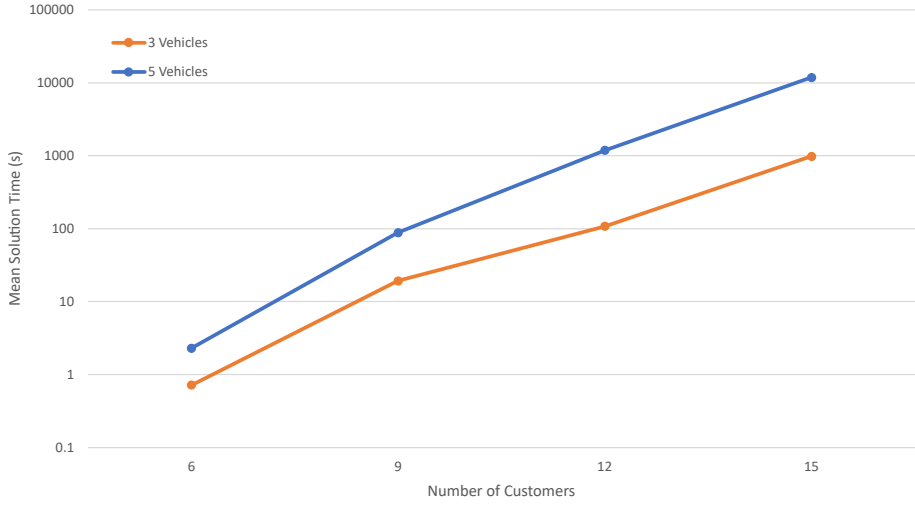

Figure 3: The mean exact solution times for model instances of varying dimensions as measured (in seconds) on a 3.41 GHz Intel i5-7500 processor with 8 GB of RAM running in a Windows 10 operating system.

Note that the vertical axis of the graph in Figure 3 is equipped with a logarithmic scale. The solution time durations therefore appear to follow an approximately linear upward trend, which very clearly demonstrates that the computational complexity of the model grows exponentially as the number of customers increases, for a fixed number of vehicles, while the computational complexity of the model also increases as the number of vehicles increases. Hence an exact solution of the model would not seem to be scalable to real-world problem instances which easily involve 300 customers and as many as 40 vehicles. This demonstrates the need for a metaheuristic solution approach that can provide reasonable approximate model solutions in polynomial time.

4 Approximate model solution procedure

As motivated in §2, we elected to solve the realistically sized instances of the CVRPTW model of the previous section by means of the state-of-the-art HGSADC algorithm. The algorithm can reportedly handle very large and/or very constrained problems exhibiting a variety of attributes without requiring significant modifications. Vidal *et al.* [39], in fact, used the algorithm to solve CVRPTW instances exhibiting both multi-depot and

multi-period attributes. This section is devoted to a description of the working of the metaheuristic in §4.1, an overview of our implementation of the algorithm in §4.2 and a discussion in §4.3 on how this implementation was verified.

4.1 Working of the metaheuristic

The high-level working of the HGSADC algorithm is summarised in pseudo-code form in Algorithm 1. The algorithm employs a number of parameters, including an allowable number ItNI of non- improving iterations, the maximum run time Tmax allocated, the probability Prep that an infeasible solution is repaired, an allowable number Itdiv of non-improving iterations that may elapse before diversification is initiated, and an allowable number Itdec of iterations before decomposition is performed.

Algorithm 1: Pseudo-code description of the HGSADC algorithm by Vidal *et al.* [39].

```
Initialise population of candidate solutions;
while number of iterations without improvement < ItNI, and time < Tmax do
    Select parent solutions P1 and P2;
    Create an offspring C via crossover with P1 and P2;
    Educate C by applying a local search procedure;
    if C infeasible then
        Insert C into infeasible sub-population;
        Repair with probability Prep;
    if C feasible then
        Insert C into feasible sub-population;
    if maximum sub-population size reached then
        Select survivors and cull sub-population;
    if best solution not improved for Itdiv iterations then
        Diversify population;
    if number of iterations (mod 100) = 0 then
        Adjust infeasibility penalty parameters;
    if more than 120 customers and (number of iterations (mod Itdec) = 0) then
        Decompose the master problem;
        Use HGSADC to solve each sub-problem;
        Reconstitute three solutions, and insert them into the population;
Return best feasible solution;
```

The algorithm allows for the consideration of "solutions" that are infeasible with respect to load, duration and time window constraints. Such infeasible solutions are, however, penalised in the objective function. Load and duration penalties are increasing functions of the cargo a vehicle carries or time it drives beyond a predetermined maximum duration. Time windows are penalised based on the amount of "time warp" a vehicle has to perform. Time warp is the opposite of waiting time. If a vehicle arrives at a customer after the end of the customer's service window, it has to "warp" back to the end of the service window. The time between when the vehicle arrives at the customer and the end of the customer's service window determines the penalty in this case. For the remainder of that route, the

vehicle is assumed to have arrived at the customer at the end of the service window.

Each candidate solution is encoded as a set of chromosomes. The number of chromosomes depends on the attributes of the CVRPTW instance under consideration. Vidal *et al.* [38, 39] included three chromosomes, namely period, depot and giant tour chromosomes. The period chromosome contains a list of integers for each customer, representing time periods during which the customers are serviced. The depot chromosome contains an integer for each customer representing the depot from which that customer is serviced. Further secondary chromosomes may be added to these in order to represent more attributes, or these chromosomes can be removed if they are not necessary; the algorithm is flexible in this respect.

The giant tour chromosome is the most consequential. All the other chromosomes may be reconstructed from this one. For each combination of period, depot and other problem attributes, the giant tour chromosome contains a sequence of values representing the order of customer visitation. Individual tours contain customers serviced by multiple vehicles, with no indication of which parts of these tours are serviced by which individual vehicles. In order to evaluate the cost of a candidate solution, the tour has to be partitioned into routes. A split algorithm was proposed by Vidal *et al.* [40] for this purpose (*i.e.,* to find optimal routes by splitting a giant tour into sub-tours for the available vehicles).

The split algorithm employed in both the 2012 and 2013 variations of the algorithm is that of Chu *et al.* [3]. This split algorithm has an $\mathcal{O}(\ell^2)$ worst-case time complexity, where ℓ denotes the number of vertices in the tour. In a technical note published in 2016, however, Vidal [36] presented a new algorithm that can split CVRPTW tours in $\mathcal{O}(\ell)$ time for unlimited vehicle fleets, or in $\mathcal{O}(\ell n)$ time if the vehicle fleet contains n vehicles. The split algorithm for a limited fleet is presented in pseudo-code form in Algorithm 2.

An $m \times n$ matrix is maintained whose entry in row k and column t is denoted by $\mathsf{potential}[k, t]$ and represents the expected cost of servicing all customers from the start of a tour until customer t has been serviced, if k vehicles are utilised. The function

$$f(i,j) = \begin{cases} \mathsf{potential}[i,j] + \mathsf{cost}(i,j) & \text{if } \mathsf{sumLoad}[j] - \mathsf{sumLoad}[i] \leq \mathsf{capacity} \\ \infty & \text{otherwise} \end{cases}$$

is used to add the cost $\mathsf{cost}(i, j)$ of travelling from customer i to customer j to the potential cost array only if this does not violate the capacity constraint of the vehicle in question. Otherwise, the cost is considered infinite, which leads the algorithm to assign the next stop in the tour to another vehicle. The quantity $\mathsf{sumLoad}[x]$ in the function $f(i, j)$ above denotes the vehicle cargo after having visited customer x.

Another $m \times n$ matrix is maintained whose entry in row k and column i is denoted by $\mathsf{pred}[k, i]$ and represents the customer at the end of the sub-tour immediately preceding the one in which customer i is included, when k vehicles are used. A domination function is employed to compare the cost of moving to the next customer from two different customers and returns True only if customer i is better than customer j as a predecessor.

Once the split algorithm has been executed, the optimal number k of vehicles can be found by comparing the cost of service values found in $\mathsf{potential}[k, N_{\text{customers}} - 1]$. Once

Algorithm 2: Pseudo-code description of the Split algorithm by Chu *et al.* [3].

```
for k = 1 to m do
    for t = 0 to n do
        potential[k, t] ← ∞;
potential[0, 0] ← 0;
for k = 0 to m − 1 do
    Clear list Λ;
    Add k to Λ;
    for t = k + 1 to n do
        if Λ_length = 0 then
            Break loop;
        potential[k + 1, t] ← potential[k, Λ_front] + f(Λ_front, t);
        pred[k + 1, t] ← Λ_front;
        if t < n then
            if not dominates(k, Λ_back, t) then
                while Λ_length > 0 and dominates(k, t, Λ_back) do
                    Pop Λ_back;
                Push t to the back of Λ;
            while Λ_length > 0 and sumLoad[t + 1] − sumLoad[Λ_front] > capacity do
                Pop Λ_front;
```

this number has been found, the final routes are determined by tracing backwards through the predecessor array $\mathsf{pred}[k, i]$.

In order to evaluate any individual solution P, the HGSADC algorithm employs the biased fitness measure

$$\mathsf{biasedFitness}(P) = \mathsf{costRank}(P) + \mathsf{diversityRank}(P) \times \left(1 - \frac{N_{\text{elite}}}{N_{\text{individuals}}}\right)$$

which accounts for both the solution cost and the diversity contribution of P to the population of solutions. After splitting and evaluating all solutions in the population, the individuals within this population are ranked according to their cost. Solutions are also assigned diversity rankings based on the well-known Hamming distance between the secondary chromosomes (the period and depot chromosomes). While the secondary chromosomes are not binary, the distance between two solutions is considered to increase by one for each corresponding element in the chromosomes that is not identical (*e.g.,* each customer that is not served during the same set of periods). Each solution is assigned a distance score, after which the diversity ranks are determined, where the solution incurring the largest distance receives the best rank. The impact of the diversity rank on the biased fitness is based on both the number N_{elite} of elite individuals to keep when culling a population and the number $N_{\text{individuals}}$ of individuals currently in the population. This impact is always less than the impact of the cost rank, causing the lowest-cost solutions always to be retained from one generation to the next.

The algorithm initialises its solution population by randomly generating a number of individuals equal to four times a pre-specified minimum sub-population size. The algorithm

Algorithm 3: Pseudo-code description of the PIX operator of Vidal *et al.* [38].

STEP 0: INHERITANCE RULE;
$td \leftarrow$ number of depot and period combinations;
Pick two random numbers between 0 and td according to a uniform distribution. Let n_1 and n_2 be respectively the smallest and largest of these numbers;
Randomly select n_1 (depot, period) couples to form the set Λ_1;
Randomly select $n_2 - n_1$ remaining couples to form the set Λ_2;
The remaining $td - n_2$ couples make up the set Λ_{mix};

STEP 1: INHERIT DATA FROM P_1;
for *each (depot, period)* (d, p)*, belonging to sets* Λ_1 *and* Λ_{mix} **do**
    Λ_1: Copy the sequence of customer visits from $V_{d,p}(P_1)$ to $V_{d,p}(C)$;
    Λ_{mix}: Randomly select two chromosome-cutting points α and β (according to a uniform distribution), then copy the α to β sub-sequence of $V_{d,p}(P_1)$ to $V_{d,p}(C)$;

STEP 2: INHERIT DATA FROM P_2;
for *each (depot, period)* $(d, p) \in \Lambda_2 \cup \Lambda_{mix}$ *selected in random order* **do**
    Consider each customer visit i in $V_{d,p}(P_2)$ and copy it to the end of $V_{d,p}(C)$ if the following two conditions are met;
    1) The depot choice $\delta_i(C)$ for the child C is equal to d or undefined (meaning no visit to i has been copied to C yet);
    2) One visit pattern of customer i, at least, contains the set $\pi_i(C) \cup p$ of visit periods;

STEP 3: COMPLETE CUSTOMER SERVICES;
Perform the *Split* algorithm and extract the routes for each (depot, period) pair;
if *the service-frequency requirements are satisfied for all customers* **then**
    Stop;
else
    while *customers with unsatisfied service-frequency requirements exist* **do**
        Randomly select a customer i for which service-frequency requirements are not satisfied;
        Let $\mathcal{F}$ be the set of admissible (depot, period) combinations (d, p) with respect to its pattern list L_i and the visits already included in C. Let $\psi(i, d, p)$ be the minimum penalised cost for the insertion of customer i from depot d in period p;
        Insert i into $(o^*, l^*) = \arg\min_{(d,p)\in\mathcal{F}}\{\psi(i, d, p)\}$;

performs several steps during each iteration. The first step involves selecting two parent solutions. Two pairs of solutions are chosen randomly from the pool of solutions currently within both the feasible set and the infeasible set. The solutions in each pair are compared, and the one with the better biased fitness value is selected as a parent.

A child solution is created by applying a crossover operator to the giant tour chromosome with the aid of the information contained in the secondary chromosomes. The 2012 paper by Vidal *et al.* [38] contains a new *periodic crossover with insertions* (PIX) operator, for which a pseudo-code description is provided in Algorithm 3. This operator pursues a balance between search space exploration and attaining improvements on existing solutions. An offspring solution inheriting similar amounts of genetic material from both parents will be substantially different from both of them, while one that inherits mostly from one parent will differ only slightly from that parent. The PIX procedure employs randomness to determine how much material to inherit from each parent instead of following predefined rules by which offspring inherit a fixed amount of material from each parent.

After a child solution has been generated, there is a chance to educate it, based on an education probability parameter. The education procedure is what differentiates the HGSADC algorithm from standard genetic algorithms, as the procedure applies a limited local search to the solutions generated in order to find quick and easy improvements. Education makes use of two local search procedures: *Route improvement* (RI) and *pattern improvement* (PI). These procedures are applied in the order RI, PI, RI.

The RI procedure seeks to optimise each tour separately. It considers various insertion, swap and 2-opt moves [7]. For each vertex in a route, a granularity threshold is employed to restrict the search neighbourhood to the closest vertices [34]. The procedure randomly iterates over each vertex and each of its neighbours, attempting to apply the moves in a random order. The procedure terminates when all possible moves have been attempted in respect of all neighbours of the current vertex without uncovering an improving move.

The PI procedure attempts to move customers between depots and periods. Iterating over each customer in a random order, it computes the minimum cost of satisfying the service requirements of the customer from the current depot and within the current period, and again with each other possible combination of depot and period. If there is another combination that yields a lower minimum cost of serving the customer, the customer is inserted in the position that achieves the minimum cost.

After having implemented the RI, PI, RI education procedure, the child solution may or may not be feasible. It is inserted into the appropriate sub-population. Based on a repair probability, infeasible solutions may be repaired. In order to repair an individual, the penalty parameters are multiplied by 10 and the education procedure is performed again. If the individual is still infeasible, the penalty parameters are multiplied by 100 and the procedure is repeated. If a feasible individual is produced by the repair procedure, this individual is added to the feasible population without modifying or removing the original infeasible individual from the infeasible sub-population.

When either sub-population reaches the maximum allowed size, it is culled down to the minimum population size. All solutions are ranked according to their biased fitness values based on both cost and diversity contributions. A list of clones is compiled, where

individuals are considered clones if either their depot and period chromosomes or their solution costs match. The clone with the worst biased fitness value is removed until there are either no more clones, or until the minimum population size is reached. If there are no clones, or all clones have been removed, the remaining individuals with the worst biased fitness values are repeatedly removed until the minimum population size is reached.

Every 100 iterations, the capacity and duration penalty parameters are adjusted, based on the proportions of naturally-feasible individuals with respect to vehicle capacity and route duration over the last 100 iterations. The parameters are adjusted slightly in order to achieve a target proportion of naturally-feasible individuals.

If more than ltdiv iterations have been performed without encountering any improving solution, both sub-populations are culled to one third of the normal minimum size. A number of new solutions equal to four times the minimum population size are then generated randomly and added to the appropriate sub-populations. This introduces a large number of novel solutions to the populations.

The main contribution of the 2013 paper by Vidal *et al.* [39] is the algorithm's decomposition procedure. It is performed once every ltdec iterations, and only on problem instances containing more than 120 customers. A random individual is selected from the 25% best feasible solutions and split into smaller parts, thus inducing sub-problems. Each tour in this individual's giant tour chromosome is iterated across sequentially, generating a new set of tours. Every time the set reaches more than 120 customers, or all tours have been visited, a new sub-problem is created from this set of tours and solved by performing another instance of the HGSADC algorithm, with the maximum non-improving iterations parameter set to ltdec/2. Once every sub-problem has been solved, the top three solutions from each sub-problem are combined into three new solutions which are then introduced into the appropriate main populations.

Finally, once the algorithm has performed ltNI iterations during which no new best solution update has occurred or if the specified maximum run time is reached, it returns the best solution encountered during the entire search.

4.2 Metaheuristic implementation

We implemented the HGSADC algorithm described in the previous section in Python 3.8, using the CPython interpreter. A handful of publicly available libraries were installed on the interpreter, including OpenPyXl and NumPy which provide useful tools for scientific computing and provided us with the capability of reading input from and writing output to Microsoft Excel files. The sequence of steps followed within each algorithmic iteration are almost identical to those described in Algorithm 1, with the addition of programming logic facilitating a comparison between generated solutions and the currently best-known solution. We adopted an object-oriented approach in order to achieve modularity and clarity of implementation.

We omitted the chromosomes accounting for the multi-period and multi-depot attributes in our implementation, replacing them with one capable of indicating which type of vehicles from the DC's heterogeneous delivery fleet should service each customer. The tours within the giant tour chromosome were therefore appropriately indexed by vehicle types, rather

than by depots and periods. The capacity, cost per kilometre and cost per hour incurred by any vehicle that services customers is determined by its type. The implementation also accounts for the number of vehicles of each type that are available for service according to the input data as well as for the possibility of split deliveries, which often occur at the SPAR Western Cape DC. This occurs when a customer exhibits a larger demand than can be satisfied by a single vehicle, in which case multiple vehicles are required to service the customer.

The customer completion procedure of Algorithm 3 was found to be insufficient for our specific context, as it was designed only to check that customers are visited during as many periods as required by the customer. A wholly new customer completion procedure, described in pseudo-code form in Algorithm 4, was therefore designed and implemented to ensure that all customers' demands are met exactly without disrupting the relative positions of split deliveries within their tours.

The split algorithm of Vidal [36] proved the most significant challenge to implement. Fortunately, Vidal made his C++ implementation of the algorithm available on GitHub [37]. This, together with a slightly different implementation by Juppe [16], provided enough guidance for us to implement the algorithm anew. In our implementation, the algorithm takes as input single tours and vehicle type information, and then returns as output lists of vehicle routes, where each route is a sequence of customers to be visited by a single vehicle.

Another aspect of note about our implementation is the way in which departure times are calculated. Although Vidal *et al.* [39] described how time-warps may be used to penalise solutions that involve vehicles arriving at customers after the end of their time windows, they did not describe how to determine the corresponding vehicle departure times. Although algorithms are described in the literature for determining optimal departure times (see, for example, [18]), these algorithms were deemed computationally too expensive to integrate with the HGSADC algorithm. A greedy heuristic was therefore designed and implemented in which vehicles are assumed to service customers at the latest possible time. Each vehicle's departure time from the depot is set such that it is expected to arrive at the first customer with a time window along its route at the end time of this customer's time window. This may result in a number of slightly sub-optimal departure times, and would require SPAR schedulers who use the model to determine the departure times for vehicles along routes without time windows themselves.

4.3 Implementation verification

A careful verification of the metaheuristic implementation described in the previous section is important, as it provides an indication of whether the solutions produced by the metaheuristic are of a sufficient quality to be recommended to the SPAR Western Cape DC. The exact CPLEX model solution implementation of §3.2 was employed as reference point in order to verify the aforementioned implementation of the metaheuristic.

Recall that during the verification of the exact model solution procedure in §3.2, randomised test instances involving small numbers of customers and vehicles were generated. The input and output data of these test instances were recorded. A script was implemented

Algorithm 4: The customer completion procedure, designed to ensure that all customers' demands are met exactly without disrupting the positions of split deliveries within the tour.

```
redistributeService ← True;
for vehicleType in vehicleTypes do
    for stop in tour do
        Find customer being serviced at stop;
        if stop.servicedDemand = 0 then
            Remove stop from tour;
            Skip to next iteration;
        else if stop.servicedDemand > vehicleType.palletCapacity then
            stop.servicedDemand ← pallet capacity;
        else if stop.servicedDemand < vehicleType.palletCapacity then
            customer.excessCapacity ← customer.excessCapacity +
              vehicleType.palletCapacity − stop.servicedDemand;
        customer.servicedDemand ← customer.servicedDemand +
          stop.servicedDemand;
if not all customers' demands are met exactly then
    while redistributeService do
        redistributeService ← False;
        Shuffle vehicle types;
        for vehicleType in vehicleTypes do
            for stop in tour do
                Find customer being serviced at stop;
                otherExcessCapacity = customer.excessCapacity −
                  (vehicleType.palletCapacity − stop.servicedDemand);
                if otherExcessCapacity > stop.servicedDemand then
                    Remove the stop from the tour;
                    redistributeService ← True;
                    Skip to next iteration;
                remainingDemand ← customer.demand - customer.servicedDemand;
                if remainingDemand > 0 then
                    Increase the demand serviced at this stop by as much as necessary
                      or possible;
                else if remainingDemand < 0 then
                    if remainingDemand + stop.servicedDemand > 0 then
                        Reduce the demand serviced at this stop by as much as
                          necessary;
                    else
                        Remove the stop from the tour;
                        Skip to next iteration;
if not all customers' demands are met exactly then
    foreach customer do
        while customer has unserviced demand do
            Select a random vehicleType;
            Create a new stop at a random index in this vehicleType's sub-tour,
              servicing as much demand as necessary or possible;
```

that would read these recorded data and sequentially execute the metaheuristic implementation of §4.2 for each instance. The metaheuristic was subjected to a maximum allowable number of 5 000 non-improving iterations (no maximum permissible run time was imposed).

The metaheuristic was verified by comparing the cost of its best solution for each test instance with the cost of an exact solution to the same instance. Occasionally the two solution approaches resulted in slightly different evaluations of the same solution. This occurred in only one of sixty of the random test instances involving 6–12 customers, but in all twenty test instances involving 15 customers, and is due to the fact that the mathematical model of §3.1 optimises each vehicle's departure time (when solved exactly). The metaheuristic of §4.1, however, adopts a greedy heuristic to guesstimate a good departure time of the vehicles, as described in the previous section. This may possibly degrade the (accidental) optimality of some solutions returned by the metaheuristic, but such degradations were observed to have a negligible impact in the case of the large problem instances for which it was intended.

In order to facilitate a fairer comparison between the solution costs for verification purposes, costs associated with the exact solutions to the mathematical model were subsequently also evaluated by means of the heuristic employed in the metaheuristic implementation of §4.2. Table 4 contains the mean differences in cost between the solutions returned by the metaheuristic and the corresponding exact solutions for the randomised test instances of §3.2. The mean difference between the solution costs and the number of test instances for which the metaheuristic produced identical solutions to those returned by CPLEX are also shown in the table. Solutions were considered identical for practical purposes when the difference in cost, as evaluated by the metaheuristic, was less than 0.001%. The difference in cost between solutions was calculated as (Meta Cost – Exact Cost)/(Exact Cost).

Number of Vehicles	Number of Customers	ΔCost	Number Identical
3	6	0.50%	90%
3	9	0.00%	100%
3	12	0.53%	80%
3	15	1.14%	70%
5	6	0.25%	80%
5	9	0.63%	70%
5	12	1.45%	50%
5	15	0.99%	50%

Table 4: Mean differences in objective function values (ΔCost) between the exact solutions returned by CPLEX and the approximate solutions returned by the metaheuristic for the 80 small test instances of §3.2.

The metaheuristic obtained 59 solutions that are identical to those returned by CPLEX out of the 80 small random test instances considered, with an overall average optimality gap of 0.56%. The metaheuristic is therefore considered to have been verified successfully.

Figure 4 contains a plot of the mean solution times incurred by the metaheuristic for these test instances on the same processor as that used in §3.2. The mean solution times incurred

by CPLEX are also plotted by means of dashed lines in the same figure for comparison purposes. When comparing the mean solution times of the two solution implementations, it is worth noting that CPLEX utilises multiprocessing, while the metaheuristic implementation only utilises a single thread. CPLEX can therefore take advantage of all the available processing power, while the metaheuristic can only use a maximum of a quarter of the processing power available on the quad-core CPU upon which the test instances were solved. Additionally, the exact solution times of the instances fluctuated to a far greater degree than did the approximate solution times. CPLEX ran so long for some of the larger test instances that the branch-and-cut procedure either exhausted all the computer memory or was interrupted by Windows updates after many hours. Such instances were re-randomised and re-run in an attempt at ensuring a fairer comparison.

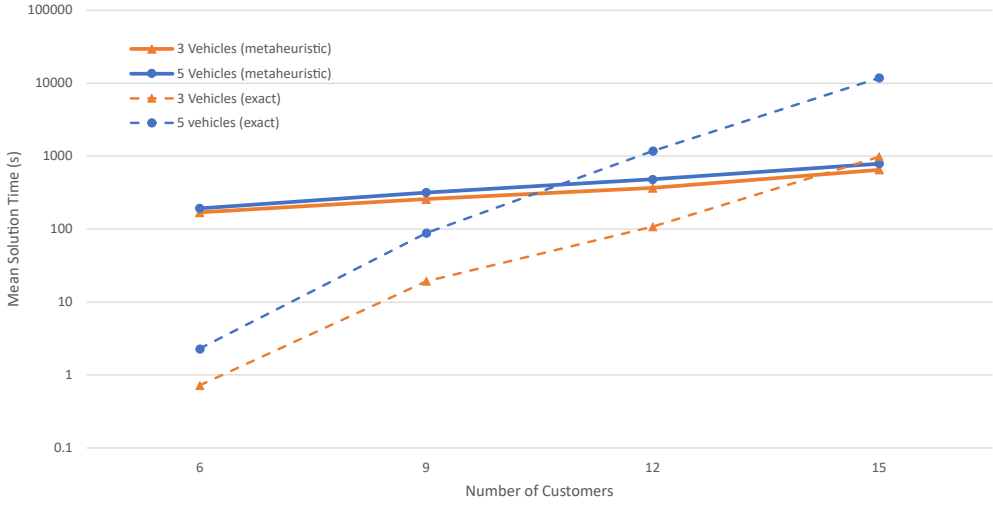

Figure 4: The mean exact and metaheuristic solution time for model instances of varying sizes as measured (in seconds) on a 3.41 GHz Intel i5-7500 processor with 8 GB of RAM running in a Windows 10 operating system.

Although the computation time required by the metaheuristic (to find approximate model solutions) would seem large in comparison with that required by CPLEX (to find exact model solutions) in Figure 4, the slope of increase in the former times (the solid line graphs in the figure) are not as steep as that of the latter times (the broken line graphs in the figure). Moreover, it is not visible in Figure 4, but the solution times required by CPLEX rise substantially as the number of customers increases beyond fifteen. In fact, for eighteen customers, CPLEX could not obtain exact solutions within 100 000 seconds (27.78 hours). In contrast, the solution times incurred by the metaheuristic are considered to be acceptable, even for relatively large problem instances.

5 Case study

The efficacy of the mathematical model of §3 and the approximate solution methodology of §4 are evaluated in this section in the form of a case study involving real data obtained

from the SPAR Western Cape DC for three work days during 2019. This case study allows for a comparative evaluation of the vehicle routes recommended by the model with the *status quo*. The section opens in §5.1 with a background discussion on the data pertaining to the case study and a discussion in §5.2 on the assumptions made in order to implement the model of §3 and the metaheuristic of §4 in the context of real data. The results returned by the metaheuristic for the three days in question are presented and discussed in some detail in §5.3.

5.1 Background information and input data

As mentioned in the introduction, the SPAR Western Cape DC serves more than 300 customers in the Western Cape and Northern Cape provinces of South Africa. Human schedulers construct daily vehicle routing schedules by hand, aiming to satisfy all customer demand on any given day. These schedulers utilise a predetermined delivery schedule to guide their routing decisions. This schedule is based on one-hour delivery windows starting at different times of the day, which are distinct for different days of the week, because some customers only specify delivery time windows on certain days of the week. Past experience of the schedulers in conjunction with this schedule allows for the construction of high-quality delivery vehicle routes that enable the DC to operate effectively in a relatively cost-efficient manner.

The DC has more than 40 vehicles in its active delivery fleet, most of which consist of interchangeable horses and trailers of varying capacities, but the DC also temporarily hires third-party vehicles and drivers, as required, in order to facilitate a flexible service capacity. Vehicles that return to the depot upon completing delivery routes may be assigned further routes to service on the same day.

All deliveries are scheduled on a central Google Sheets document visible to multiple parties within the DC. Once a vehicle has departed from the DC, information about the vehicle's route is removed from the live document and stored in an archive. The 2019 archive shows that the number of deliveries scheduled per day varied between 2 and 369, with a mean of 169 (190 when excluding Sundays). These numbers allowed us to identify three days — a relatively quiet day, an average day, and a busy day — on which to base the case study of this section. Information pertaining to the numbers of deliveries, delivery locations, vehicle routes, in-house and third-party vehicles, as well as demand volumes, experienced on these three days are shown in Table 5.

Day	Day type	Number of Deliveries	Number of Locations	Number of Routes	Demand (pallets)	Active vehicles	3rd-Party vehicles
7 Oct	Quiet	136	79	62	1 440	30	16
30 Oct	Average	212	123	86	2 105	38	18
26 Nov	Busy	287	136	128	3 310	36	33

Table 5: Archived dispatch data of the SPAR Western Cape DC for three days in 2019.

The archived data for the three days listed in Table 5 were formatted into the appropriate format required by our metaheuristic implementation in order to facilitate an evaluation of the performance of the model of §3 and the metaheuristic of §4 in the context of real data.

For the sake of transparency and reproducibility, these data are available electronically [30].

The vehicle type information summarised in Table 6 was valid for all three days considered in this case study. The values for the cost per kilometre travelled and the cost per hour expended by these vehicles are specified in Rands in the table and are similar to the real values provided by SPAR, but deliberately rounded. For these values, the ratio of the cost per kilometre to the cost per hour results in the cost of distance travelled dominating vehicle routing decisions.

Vehicle Type	Cost/km (R)	Cost/hour (R)	Capacity (pallets)	Number Owned	Hired cost Multiplier
Rigid	25	60	16	18	1.25
8 Metre	30	60	22	7	1.25
11 Metre	35	60	30	32	1.25
Link	40	60	40	1	1.50

Table 6: Information pertaining to the active vehicle fleet of the SPAR Western Cape DC.

5.2 Simplifying assumptions

The model of §3 does not, of course, represent all the complexities of the real-world situation exactly, and so our most important model simplifications are addressed in this section in order to facilitate a meaningful evaluation of its efficacy.

Each of the three problem instances (days) considered in the case study covers a period of time large enough for vehicles to service multiple routes upon having returned to the DC in order to load new pallets and having been assigned new drivers. These nuances were excluded from the scope of the small test problem instances considered in §3.2 and §4.3, but are accounted for during the evaluation process documented here.

As mentioned, the DC has the option of hiring additional vehicles in order to complement its own fleet, as necessary. The linear-time unlimited-fleet split algorithm [36] was therefore used when solving the three problem instances considered in this section, as opposed to the polynomial-time limited-fleet split algorithm used during the verification performed in §4.3. This achieved a significant decrease in the required solution time for the larger two problem instances.

The evaluation of the cost associated with solutions returned by the metaheuristic was also configured to estimate the number of vehicles required by summing the expected durations of routes together with additional "wasted" time between routes, and dividing this sum by a standard number of hours of service per vehicle per day. Face validation of the cost estimation thus produced was performed by February [10], the transport controller at the SPAR Western Cape DC at the time of writing. He is responsible for research into the costing of the DC's deliveries and routing decisions, and thoroughly understands the method of cost estimation utilised at the DC. He confirmed that the cost estimations returned by the method described above were close enough to the estimates produced by the DC's ERP systems to facilitate meaningful comparisons.

In reality, hiring vehicles is often cheaper for the DC than using its own vehicles, largely

due to how well the DC pays its drivers. It is, however, still preferable for the DC to use its own vehicles and drivers before hiring third-party resources. A cost multiplier was therefore applied to any vehicles used beyond the DC's own active delivery fleet. It is assumed that the first routes appearing in any recommended tour will be assigned to the DC's own vehicles, until all of these vehicles have been exhausted, upon which the cost multiplier is applied to later routes that exceed the total service time estimated to be available to the DC's own vehicle fleet.

The total loads of routes in the DC archive often appear to be greater than the capacities of the vehicles assigned to routes. This is feasible in reality if some of the pallets destined for customers along the vehicle's delivery route are not completely full. The employees loading the vehicles are able to consolidate these partially full pallets in a bid to reduce the total pallet count. In contrast, the model has no way of determining how full pallets are. It would not, however, be sensible, when evaluating the performance of the model, to assume that vehicles have soft load capacities. The demands of customers along such a route were instead reduced slightly until the vehicle's capacity was no longer exceeded. Care was taken not to reduce any customer's demand to zero, thus ensuring that any feasible solution would still visit all customers.

Sometimes distinct customers — such as a TOPS and a SUPERSPAR — would occupy the same physical location, but are nevertheless treated as separate customers by the DC's ERP systems. In such cases, the schedulers will, of course, schedule stops at these customers to occur immediately adjacent to one another along a delivery vehicle route. In order not to increase the problem instance dimensions unnecessarily, demand and consecutive stops for customers at the same location were merged when performing the case study of this section. The values in the Locations column in Table 5 represent these merged customer locations.

The DC's geo-tracking system was able to provide average unload times per pallet for most locations, but not all. For locations for which unload times were not available, unload times were assumed to be 5.5 minutes per pallet, on average.

We retrieved the expected travel distances and travel times between customers using the Bing Maps API [20]. Since no vehicle departure or arrival times were available prior to solving the model, the results requested from the API conform to the expected average travel times between the locations. In addition, 45 minutes of rest time were added to the durations of routes for every five hours of continuous driving. Eight hours of rest time were added to the duration of routes for every ten hours of total driving time required.

Since the case study of this section is aimed at ascertaining the potential cost savings of the DC when using the model proposed in this paper to guide its delivery vehicle routing decisions, the usual customer service time windows were ignored. As mentioned, these scheduled delivery windows merely serve to guide the human schedulers' routing decisions. We believe that greater cost improvements can be realised by relaxing these windows.

5.3 Numerical results

Despite employing the linear-time split algorithm, our metaheuristic implementation was found to execute slower than expected when solving the three case study problem instances. It created, educated and evaluated approximately 100 solutions every two minutes on a 3.41 GHz Intel i5-7500 processor with 8 GB of RAM running in a Windows 10 operating system. The model was initialised with parameter values specifying that at most 2 500 non-improving iterations and at most 7 200 seconds of run time should be allowed. The thresholds for performing a problem decomposition phase were reduced to 1 000 non-improving iterations and 60 customers.

In order to take advantage of the DC schedulers' experience and knowledge, the algorithm was seeded with a solution derived from the DC's archived data in addition to the population of random solutions it generates upon initialisation. This seeded solution was used during parent selection and was iteratively improved alongside the novel randomly generated solutions.

The vehicle routes actually implemented at the DC during the high-demand day (26 November 2019) are tabulated in Tables 11–13 in the appendix at the end of the paper. In total, there were 211 stops along 128 routes in this set of vehicle routes. For each route, the anonymised store names are provided in the order visited by the vehicle upon leaving the DC and before returning to the DC, together with the number of pallets delivered at each location along the route in brackets, as well as the total travel distance, travel time, and service cost associated with the route. The route costs tabulated are as they appear before having applied the multiplier for hired vehicles. The total estimated cost of implementing these routes was R 1 564 771.87.

The *status quo* solution in Tables 11–13 was included in the initial population of solutions with which the algorithm of §4 was initialised. The vehicle routes appearing in the best solution returned upon termination of the algorithm for the high-demand day are summarised in Tables 7–9 in the same format as that used in the appendix. In total, there are 170 stops along 107 routes in the vehicle route set returned by the algorithm, and the total anticipated cost of implementing these routes is R 1 410 468.59. This represents a R 169 522.45 cost saving, which equates to 10.83%, over the route set actually implemented.

The structure of the solution in Tables 7–9 suggests that the DC should have relied more heavily on its larger vehicles to perform deliveries on the day under consideration, according to the metaheuristic. The consolidation of multiple smaller loads into larger loads allows for a large reduction in cost, as it reduces the frequency with which the depot must be visited. The solutions returned by the metaheuristic, in fact, exhibit an increasing reliance on the smaller vehicle types as demand increases and the cost of hiring additional vehicles becomes more significant.

The numbers of stops and routes, as well as the estimated total costs, of the *status quo* and the metaheuristic-recommended route sets are provided in Table 10 for all three days considered in this case study. As is clear from the difference columns of the table, the estimated costs associated with solutions returned by the metaheuristic of §4 are significantly better than those associated with the actual archived route sets implemented.

Delivery (capacity)	**Route** ..., **Customer(load)**, ...	**Distance (kilometres)**	**Time (hours)**	**Cost (Rands)**
1 (16)	186-V(8)	13.65	1.23	414.75
2 (16)	45-D(13), 11-B(3)	284.3	4.44	7 373.72
3 (16)	29-C(16)	40.69	1.46	1 105.08
4 (16)	100-L(15)	276.33	4.35	7 169.10
5 (22)	130-P(19)	56.98	2.30	1 847.25
6 (22)	207-Y(22)	195.2	4.43	6,121.59
7 (22)	76-H(20), 61-G(2)	287.64	5.23	8 942.75
8 (22)	134-P(21)	25.04	2.01	871.57
9 (22)	185-V(22)	186.7	4.47	5 869.22
10 (22)	172-S(20)	417.83	5.28	12 851.41
11 (22)	29-C(22)	40.69	1.75	1 325.84
12 (22)	89-K(22)	1 584.54	25.72	49 079.70
13 (30)	125-P(14), 132-P(15)	140.95	4.45	5 200.32
14 (30)	101-L(21), 195-W(1)	85.46	2.94	3 167.29
15 (30)	91-K(22), 81-H(5), 112-M(2)	64.34	4.03	2 493.50
16 (30)	135-P(30)	43.81	2.74	1 697.46
17 (30)	69-G(30)	19.13	2.72	832.80
18 (30)	85-K(30)	1 668.16	27.56	60 038.99
19 (30)	143-P(30)	60.95	3.66	2 353.10
20 (30)	198-W(13)	24.18	2.21	978.56
21 (30)	176-T(8), 35-C(21)	294.02	5.46	10 618.17
22 (30)	93-K(10), 151-R(18)	42.61	3.22	1 684.30
23 (30)	152-R(4), 104-M(26)	146.83	4.64	5 417.51
24 (30)	156-S(30)	71.41	3.66	2 719.14
25 (30)	98-L(30)	319.10	5.10	11 474.50
26 (30)	84-K(30)	1 713.54	26.92	61 589.38
27 (30)	1-A(11), 72-G(15)	37.62	5.06	1 620.59
28 (30)	131-P(7), 95-K(18)	65.12	3.22	2 472.31
29 (30)	17-B(16), 19-B(14)	95.07	2.99	3 506.92
30 (30)	9-B(11), 8-B(5), 33-C(10)	55.84	3.45	2 161.67
31 (30)	69-G(30)	19.13	2.72	832.80
32 (30)	97-K(29)	48.73	2.56	1 859.28
33 (30)	46-D(24), 39-C(5)	72.99	3.94	2 790.88
34 (30)	96-K(29)	11.68	3.47	617.22
35 (30)	188-V(26)	76.00	3.65	2 878.94
36 (30)	182-U(17), 118-N(9), 143-P(3)	75.52	4.24	2 897.52
37 (30)	28-C(28)	62.37	3.22	2 376.26
38 (30)	22-B(13), 58-G(15)	405.56	6.50	14 584.37
39 (30)	29-C(30)	40.69	2.14	1 552.37
40 (30)	49-D(30)	79.23	2.81	2 941.90
41 (30)	91-K(28)	43.43	3.59	1 735.11
42 (30)	56-F(11), 66-G(19)	113.62	3.31	4 175.34
43 (30)	90-K(16), 200-W(13)	38.88	3.87	1 593.34
44 (30)	15-B(30)	265.88	4.35	9 566.70
45 (30)	17-B(30)	94.23	2.00	3 417.98
46 (30)	150-R(30)	315.2	5.55	11 365.19
47 (30)	18-B(24)	33.18	8.20	1 653.43
48 (30)	1-A (30)	37.42	9.13	1 857.39
49 (30)	68-G(8), 193-V(19), 75-H(3)	83.6	5.05	3 228.88
50 (30)	40-C(30)	357.60	5.34	12 836.33

Table 7: Routes recommended by the metaheuristic of §4 to the SPAR Western Cape DC for 26 November 2019.

Delivery (capacity)	**Route** ..., **Customer(load)**, ...	**Distance (kilometres)**	**Time (hours)**	**Cost (Rands)**
51 (30)	186-V(30)	13.65	3.74	702.28
52 (30)	139-P(30)	292.95	4.68	10 534.21
53 (30)	83-H(30)	55.27	3.32	2 133.77
54 (30)	158-S(30)	1 131.38	21.77	40 904.75
55 (30)	95-K(30)	67.40	3.73	2 582.75
56 (30)	92-K(30)	196.46	3.91	7 110.72
57 (30)	69-G(22), 50-D(2), 78-H(6)	29.99	3.86	1 281.44
58 (30)	12-B(1), 57-F(21), 113-M(4)	57.35	3.80	2 235.03
59 (30)	72-G(30)	28.47	3.06	1 180.11
60 (30)	13-B(16), 173-T(13)	53.24	3.75	2 087.96
61 (30)	53-E(27)	36.36	1.85	1 383.46
62 (30)	175-T(23), 206-W(7)	367.60	7.98	13 345.21
63 (30)	156-S(7), 80-H(16)	72.46	2.95	2 713.05
64 (30)	87-K(2), 155-S(1), 94-K(6), 102-L(4), 62-G(13)	82.12	6.07	3 238.54
65 (30)	24-B(9), 150-R(19)	336.68	5.63	12 121.60
66 (30)	40-C(22), 110-M(7), 60-G(1)	361.19	5.72	12 985.05
67 (30)	12-B(30)	30.67	3.08	1 258.36
68 (30)	121-O(29)	54.95	3.25	2 118.26
69 (30)	168-S(24), 7-B(4)	75.12	4.53	2 900.86
70 (30)	51-D(16), 171-S(5), 120-O(9)	77.00	5.09	3 000.72
71 (30)	174-T(14), 104-M(16)	143.91	4.42	5 301.82
72 (30)	191-V(4), 157-S(26)	68.92	3.95	2 649.12
73 (30)	124-O(30)	844.00	12.13	30 267.60
74 (30)	128-P(17), 48-D(10)	65.31	3.95	2 522.53
75 (30)	20-B(30)	1 588.94	26.59	57 208.53
76 (30)	105-M(5), 166-S(4), 115-N(7), 164-S(2), 47-D(12)	96.35	4.83	3 661.83
77 (30)	205-W(9), 144-Q(21)	224.43	5.65	8 194.12
78 (30)	69-G(30)	19.13	2.72	832.80
79 (30)	59-G(11), 92-K(16)	236.64	4.40	8 546.31
80 (30)	109-M(28), 106-M(1)	371.28	6.24	13 369.52
81 (30)	25-C(7), 111-M(16), 149-R(7)	888.36	12.26	31 828.19
82 (30)	20-B(18), 116-N(5)	1 591.25	26.03	57 255.13
83 (30)	111-M(24), 165-S(4)	812.62	10.41	29 066.08
84 (30)	147-R(7), 85-K(23)	1 668.57	27.60	60 056.00
85 (30)	109-M(30)	370.72	6.23	13 349.07
86 (30)	178-T(28)	73.50	2.89	2 745.62
87 (30)	149-R(30)	846.87	11.21	30 313.26
88 (30)	99-L(30)	647.70	9.19	23 220.91
89 (30)	142-P(26)	19.03	4.11	912.70
90 (30)	158-S(5), 84-K(22)	1 717.28	26.72	61 707.92
91 (30)	85-K(30)	1 668.16	27.56	60 038.99
92 (30)	15-B(26)	265.88	4.15	9 554.43
93 (30)	151-R(30)	15.10	2.93	704.48
94 (30)	3-A(30)	67.89	2.69	2 537.44
95 (30)	2-A(10), 119-O(12), 83-H(8)	46.60	4.24	1 885.56
96 (30)	208-Z(30)	44.60	3.27	1 757.41
97 (30)	22-B(30)	350.85	5.88	12 632.70
98 (30)	59-G(30)	213.28	4.48	7 733.41
99 (30)	135-P(2), 197-W(11), 117-N(2), 126-P(14)	155.71	4.61	5 726.29
100 (30)	163-S(18), 208-Z(11)	61.55	2.96	2 331.95

Table 8: Routes recommended by the metaheuristic of §4 to the SPAR Western Cape DC for 26 November 2019 (Continued).

Delivery (capacity)	Route ..., Customer(load), ...	Distance (kilometres)	Time (hours)	Cost (Rands)
101 (30)	26-C(20), 99-L(10)	743.50	12.57	26 776.64
102 (30)	201-W(30)	841.55	11.59	30 149.38
103 (40)	42-C(1), 153-S(33)	306.57	7.17	12 692.93
104 (40)	23-B(40)	224.41	6.02	9 337.54
105 (40)	154-S(31)	49.64	2.89	2 158.67
106 (40)	10-B(40)	371.25	7.71	15 312.71
107 (40)	41-C(39)	462.72	8.07	18 992.95

Table 9: Routes recommended by the metaheuristic of §4 to the SPAR Western Cape DC for 26 November 2019 (Continued).

The relative percentage improvements for the different days of the case study make sense when compared with one another. The second of these days, 30 October 2019, was an average day, with demand levels similar to the vast majority of days in the year. The DC's schedulers are accustomed to dealing with this level of demand and their guiding time-window schedule is focused towards handling this type of day. The first day, 7 October 2019, however experienced a lower demand than usual. The schedulers' reliance on pre-determined delivery time windows may have impacted negatively on their overall routing efficiency on this day. Finally, 26 November 2019 was part of a higher-than-average demand period just before Black Friday. When faced with such a peak in demand, mistakes and inefficiencies are amplified. For example, one customer in that day's archive appears to have been served four times, with vehicles departing along routes consisting only of visiting this customer (at 9:55, 12:10, 20:30, and 22:20). None of these vehicles carried full loads. It is not hard for the metaheuristic (with perfect information on that day's demand) to find better routing solutions.

	Actual			Metaheuristic			Cost reduction	
Day	Cost (R)	Stops	Routes	Cost (R)	Stops	Routes	(R)	(%)
7 Oct	588 019.64	91	62	541 168.94	82	49	45 774.21	7.78
30 Oct	692 146.30	144	86	661 103.96	125	71	28 354.56	4.10
26 Nov	1 564 771.87	211	128	1 410 468.59	170	107	169 522.45	10.83

Table 10: A comparison of statistics related to the delivery vehicle routes actually implemented at the SPAR Western Cape DC on three days in 2019 and the routes recommended by the metaheuristic of §4 for these days.

It is acknowledged that the three problem instances of this case study were solved outside of the busy and chaotic context of managing operations at the DC. In reality, schedulers will not have perfect information on a day's demand levels. Moreover, the expected distances and durations of travel and service are only estimates, and there will invariably be other delays and complications beyond the scope of the model of §3 to cope with. It is therefore expected that a portion of the savings indicated in Table 10 will be lost to these real-world inefficiencies when applying the metaheuristic-recommended routing decisions in practice. Nevertheless, there is clearly a potential for the SPAR Western Cape DC to realise considerable cost savings if their schedulers were to utilise the model and accompanying metaheuristic of this paper to augment daily vehicle routing decisions.

Finally, the modelling approach is rather flexible and can be applied in a great variety

of situations. It may, for example, be used to plan the servicing of known demand for a full day (or more), over the next twelve hours (the minimum time required at the DC before picking of customer orders may take place), or over a period of a few hours in advance (including only the demand serviced by one or two picking waves in the DC). Time windows can, furthermore, be customised independently for each customer, allowing for experimentation and further optimisation in cases where customers are not insistent on receiving service during specified periods of time. The maximum allowable number of non-improving iterations and the maximum run time can also be customised as required, allowing the metaheuristic to be left running for longer periods of time with a view to find better solutions, especially when exploring the search spaces of large problem instances. The algorithm can also be allowed to use an unlimited fleet of vehicles, or be forced to utilise only a limited fleet (which is only recommended for smaller problem instances, as it otherwise significantly slows down the algorithmic implementation). The costs of using different vehicle sizes, as well as the cost multipliers applied for hired vehicles, can be adjusted to influence the proportions of different vehicle sizes utilised.

6 Conclusion

In this paper, we formulated a CVRPTW model appropriate for use at the SPAR Western Cape DC, implemented this model in CPLEX, and verified the implementation in respect of a sample of small randomised test instances. As expected, however, the branch-and-cut method employed by CPLEX required solution times that increase exponentially as the problem instance dimensions increase linearly. This indicated the need for a metaheuristic approach towards solving realistically sized problem instances (approximately).

A state-of-the-art hybrid metaheuristic, called the HGSADC algorithm, was therefore implemented in Python with minor alterations (such as those applied to the customer completion procedure and the secondary chromosomes utilised). This metaheuristic implementation was again verified in the context of the same small test instances solved by CPLEX. While the metaheuristic also exhibited solution times that increase exponentially with linear increases in the model instance dimensions, the rate of increase was far smaller than that incurred by CPLEX.

A practical case study was finally performed by applying the HGSADC algorithm to historical routing data obtained from the SPAR Western Cape DC, pertaining to three workdays of representative dispatch workloads. After making a number of necessary simplifications, the costs of the historically implemented routes for these workdays were evaluated and compared with the costs associated with routing solutions recommended by the metaheuristic. Significant cost savings of the order of 5–10% were thus realised. While admittedly a simplification of reality, the cost estimation procedure employed during this comparison was face-validated by the transport controller at the DC, lending credence to the cost savings reported.

The potential benefits of utilising the metaheuristic to assist the dispatch department of the DC in routing decisions are compelling, but caution is advised in accepting the reported savings at face value. Many simplifications were necessary during the derivation of the CVRPTW model, including the implicit assumption that all input information pertaining

to a problem instance is deterministic. Execution of the recommended routing decisions in the real world may therefore well result in slight degradations of the expected savings. The reported daily cost savings are nevertheless significant and warrant consideration of an integration of commercial vehicle routing decision support software into the ERP systems already in use at the DC instead of relying entirely on manual vehicle routing.

References

[1] Barnard A, 2020, *Industrial engineer*, SPAR Western Cape DC, Personal communication, Contactable at `Andriette.Barnard@spar.co.za`.

[2] Christofides N, Mingozzi A & Toth P, 1979, *The Vehicle Routing Problem*, Wiley, Chichester.

[3] Chu F, Labadi N & Prins C, 2006, *A scatter search for the periodic capacitated arc routing problem*, European Journal of Operational Research, **169(2)**, pp. 586–605.

[4] Clarke G & Wright JW, 1964, *Scheduling of vehicles from a central depot to a number of delivery points*, Operations Research, **12(4)**, pp. 568–581.

[5] Cordeau J-F, Laporte G & Mercier A, 2001, *A unified tabu search heuristic for vehicle routing problems with time windows*, Journal of the Operational Research Society, **52(8)**, pp. 928–936.

[6] Cordeau J-F & Maischberger M, 2012, *A parallel iterated tabu search heuristic for vehicle routing problems*, Computers and Operations Research, **39(9)**, pp. 2033–2050.

[7] Croes GA, 1958, *A method for solving traveling salesman problems*, Operations Research, **6**, pp. 791–812.

[8] Dantzig GB & Ramser JH, 1959, *The truck dispatching problem*, Management Science, **6(1)**, pp. 80–91.

[9] ESRI, 2020, *Logistics and fulfillment*, [Online], [Cited December 2020], Available: `https://www.esri.com`.

[10] February N, 2020, *Transport controller*, SPAR Western Cape DC, Personal communication, Contactable at `Neil.February@spar.co.za`.

[11] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13(5)**, pp. 533–549.

[12] Golden BL, Wasil EA, Kelly JP & Chao I-M, 1998, *The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets and computational results*, pp. 33–56 in Crainic TG & Laporte G (Eds), *Fleet management and logistics*, Springer, Boston (MA).

[13] Groër C, Golden B & Wasil E, 2010, *A library of local search heuristics for the vehicle routing problem*, Mathematical Programming Computation, **2(2)**, pp. 79–101.

[14] Groër C, Golden B & Wasil E, 2011, *A parallel algorithm for the vehicle routing problem*, INFORMS Journal on Computing, **23(2)**, pp. 315–330.

[15] Jin J, Crainic TG & Løkketangen A, 2012, *A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems*, European Journal of Operational Research, **222(3)**, pp. 441–451.

[16] Juppe J, 2018, *Hybrid genetic search with adaptive diversity control*, [Online], [Cited May 2020], Available: `https://github:com/jjuppe/Hybrid-Genetic-Search-with-Adaptive-Diversity-Control`.

[17] Kendall KE & Kendall JE, 2014, *Systems Analysis and Design*, 9th Edition, Pearson, Harlow.

[18] Kok AL, Hans EW & Schutten JMJ, 2011, *Optimizing departure times in vehicle routes*, European Journal of Operational Research, **210(3)**, pp. 579–587.

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

[20] Microsoft, 2020, *Bing maps dev centre*, [Online], [Cited July 2020], Available: `https://www:bingmapsportal:com/`.

[21] Nagata Y & Bräysy O, 2009, *Edge assembly-based memetic algorithm for the capacitated vehicle routing problem*, Networks, **54(4)**, pp. 205–215.

[22] Onstein AT, Ektesaby M, Rezaei J, Tavasszy LA & van Damme DA, 2020, *Importance of factors driving firms' decisions on spatial distribution structures*, International Journal of Logistics Research and Applications, **23(1)**, pp. 24–43.

[23] OPSI Systems, 2020, *Plato*, [Online], [Cited December 2020], Available: `https://www.opsisystems.com/products/view/plato`.

[24] OptimoRoute, 2020, *Organizing the mobile workforce*, [Online], [Cited December 2020], Available: `https://optimoroute.com/`.

[25] Pisinger D & Ropke S, 2007, *A general heuristic for vehicle routing problems*, Computers and Operations Research, **34(8)**, pp. 2403–2435.

[26] Prins C, 2009, *A GRASP × evolutionary local search hybrid for the vehicle routing problem*, pp. 35–53 in Pereira FB & Tavares J (Eds), *Bio-inspired algorithms for the vehicle routing problem*, Springer, Berlin.

[27] Reimann M, Doerner K & Hartl RF, 2004, *D-ants: Savings-based ants divide and conquer the vehicle routing problem*, Computers and Operations Research, **31(4)**, pp. 563–591.

[28] Route4Me, 2020, *Route planning for your business*, [Online], [Cited December 2020], Available: `https://www.route4me.com/`.

[29] Samsara, 2020, *Streamline your operations — One platform for fleet management, driver safety, and compliance*, [Online], [Cited December 2020], Available: `https://www.samsara.com/`.

[30] Shuttleworth AD, 2020, *VRP case study*, [Online], [Cited September 2020], Available: `https://github:com/Greatmar2/VRP-Case-Study`.

[31] SPAR, 2017, *Group profile*, [Online], [Cited March 2020], Available: `https://investor-relations:spar:co:za/ir2017/who-we-are/group-profile/`.

[32] Subramanian A, Uchoa E & Ochi LS, 2013, *A hybrid algorithm for a class of vehicle routing problems*, Computers and Operations Research, **40(10)**, pp. 2519–2531.

[33] Tarantilis CD, 2005, *Solving the vehicle routing problem with adaptive memory programming methodology*, Computers and Operations Research, **32(9)**, pp. 2309–2327.

[34] Toth P & Vigo D, 2003, *The granular tabu search and its application to the vehicle routing problem*, INFORMS Journal on Computing, **15(4)**, pp. 333–346.

[35] Toth P & Vigo D, 2014, *Vehicle Routing: Problems, Methods and Applications*, 2nd Edition, SIAM, Philadelphia (PA).

[36] Vidal T, 2016, *Technical note: Split algorithm in $O(n)$ for the capacitated vehicle routing problem*, Computers and Operations Research, **69**, pp. 40–47.

[37] Vidal T, 2018, *Split library*, [Online], [Cited April 2020], Available: `https://github:com/vidalt/Split-Library`.

[38] Vidal T, Crainic TG, Gendreau M, Lahrichi N & Rei W, 2012, *A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems*, Operations Research, **60(3)**, pp. 611–624.

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

[40] Vidal T, Crainic TG, Gendreau M & Prins C, 2013, *Heuristics for multi-attribute vehicle routing problems: A survey and synthesis*, European Journal of Operational Research, **231(1)**, pp. 1–21.

[41] Zachariadis EE & Kiranoudis CT, 2010, *A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem*, Computers and Operations Research, **37(12)**, pp. 2089–2105.

Appendix

Delivery (capacity)	**Route** ..., **Customer(load)**, ...	**Distance (kilometres)**	**Time (hours)**	**Cost (Rands)**
1 (16)	7-B(4), 188-V(6), 39-C(5)	80.95	4.00	2 263.77
2 (16)	120-O(9), 198-W(6)	27.04	2.02	796.98
3 (16)	80-H(16)	66.58	2.16	1 794.08
4 (16)	69-G(15)	19.13	1.60	574.25
5 (16)	134-P(13)	25.04	1.43	711.98
6 (16)	92-K(16)	196.46	3.18	5 102.47
7 (16)	112-M(2), 198-W(7), 42-C(1)	24.36	1.99	728.71
8 (16)	51-D(12)	68.74	2.79	1 885.83
9 (16)	62-G(13), 155-S(1), 87-K(2)	68.50	4.36	1 973.98
10 (16)	47-D(12), 166-S(4)	91.24	2.78	2 448.21
11 (16)	128-P(14)	67.93	2.37	1 840.40
12 (16)	105-M(5), 46-D(4), 115-N(7)	94.85	3.62	2 588.50
13 (16)	191-V(4), 48-D(10), 96-K(2)	70.74	3.29	1 965.98
14 (16)	186-V(16)	13.65	2.14	469.68
15 (16)	134-P(8), 171-S(5)	31.06	1.99	895.92
16 (16)	93-K(10), 151-R(6)	42.61	2.21	1 197.99
17 (16)	8-B(5), 33-C(5)	37.07	1.59	1 022.11
18 (16)	142-P(4), 12-B(11)	30.20	2.24	889.66
19 (16)	130-P(3), 53-E(13)	56.64	1.94	1 532.38
20 (16)	13-B(16)	56.00	2.36	1 541.47
21 (16)	9-B(11)	50.88	1.90	1 386.16
22 (16)	97-K(15)	48.73	1.68	1 318.78
23 (16)	75-H(3), 135-P(6)	67.58	1.80	1 797.40
24 (16)	90-K(16)	28.37	1.77	815.55
25 (16)	128-P(1)	67.93	1.10	1 764.57
26 (16)	128-P(1)	67.93	1.10	1 764.57
27 (16)	128-P(1)	67.93	1.10	1 764.57
28 (16)	195-W(1)	44.28	1.05	1 170.12
29 (16)	83-H(16)	55.27	2.31	1 520.61
30 (16)	69-G(7)	19.13	1.00	538.38
31 (16)	158-S(13)	1 131.38	20.21	29 497.40
32 (22)	156-S(17), 132-P(5)	94.12	3.49	3 032.70
33 (22)	17-B(10), 164-S(1), 197-W(11)	174.27	3.86	5 459.94
34 (22)	207-Y(22)	195.20	4.43	6 121.59
35 (22)	83-H(22)	55.27	2.74	1 822.87
36 (22)	119-O(12), 2-A(10)	47.37	3.49	1 630.35
37 (22)	46-D(20)	67.98	2.72	2 202.58
38 (22)	188-V(20)	76.00	3.04	2 462.63
39 (22)	151-R(20)	15.10	2.10	578.83
40 (22)	56-F(11), 118-N(9)	97.45	3.86	3 155.15

Table 11: Routes actually implemented at the SPAR Western Cape DC on 26 November 2019.

Delivery (capacity)	Route ..., Customer(load), ...	Distance (kilometres)	Time (hours)	Cost (Rands)
41 (22)	72-G(15), 18-B(7)	33.51	4.14	1 253.67
42 (22)	58-G(15), 22-B(7)	405.80	6.09	12 539.17
43 (22)	142-P(22)	19.03	3.55	784.02
44 (22)	50-D(2), 78-H(6), 113-M(4), 94-K(6), 102-L(4)	61.40	5.31	2 160.89
45 (22)	35-C(21)	282.35	4.61	8 746.94
46 (22)	168-S(13), 17-B(7)	110.30	2.81	3 477.84
47 (22)	163-S(18), 3-A(4)	72.92	2.39	2 330.80
48 (22)	186-V(22)	13.65	2.83	579.11
49 (22)	205-W(9), 45-D(6), 206-W(7)	282.10	6.26	8 838.62
50 (22)	144-Q(21), 106-M(1)	225.06	4.45	7 018.66
51 (22)	157-S(10), 28-C(11), 11-B(1)	76.23	3.05	2 469.56
52 (22)	121-O(6), 143-P(4), 193-V(7), 185-V(5)	215.65	4.97	6 767.42
53 (22)	135-P(11), 95-K(3), 182-U(7)	76.07	2.65	2 441.17
54 (22)	151-R(22)	15.10	2.27	588.86
55 (22)	173-T(13), 131-P(7)	55.89	2.94	1 853.31
56 (22)	40-C(22)	357.60	4.90	11 022.07
57 (22)	57-F(21)	53.98	2.67	1 779.61
58 (22)	89-K(22)	1 584.54	25.72	49 079.70
59 (22)	158-S(22)	1 131.38	21.04	35 203.82
60 (22)	84-K(22)	1 713.54	26.19	52 977.67
61 (30)	59-G(30)	213.28	4.48	7 733.41
62 (30)	154-S(1), 68-G(8), 1-A(21)	49.18	8.64	2 239.48
63 (30)	72-G(30)	28.47	3.06	1 180.11
64 (30)	29-C(30)	40.69	2.14	1 552.37
65 (30)	18-B(17), 200-W(13)	36.85	7.93	1 765.51
66 (30)	96-K(27)	11.68	3.25	604.12
67 (30)	152-R(4), 15-B(26)	270.90	4.70	9 763.61
68 (30)	139-P(30)	292.95	4.68	10 534.21
69 (30)	24-B(9), 25-C(7), 59-G(11)	287.28	5.78	10 401.40
70 (30)	40-C(30)	357.60	5.34	12 836.33
71 (30)	117-N(2), 126-P(14), 125-P(14)	145.80	3.96	5 340.60
72 (30)	157-S(15), 95-K(15)	71.42	3.68	2 720.64
73 (30)	135-P(15), 33-C(5), 182-U(10)	78.40	3.30	2 941.84
74 (30)	121-O(14), 143-P(15)	65.03	3.60	2 492.33
75 (30)	66-G(19), 168-S(11)	116.14	3.33	4 264.92
76 (30)	69-G(30)	19.13	2.72	832.80
77 (30)	153-S(30)	305.34	6.49	11 076.64
78 (30)	91-K(27)	43.43	3.48	1 728.96
79 (30)	49-D(30)	79.23	2.81	2 941.90
80 (30)	208-Z(30)	44.60	3.27	1 757.41
81 (30)	29-C(10), 1-A(20)	40.44	6.90	1 829.11
82 (30)	12-B(20)	30.67	2.31	1 212.03
83 (30)	81-H(5), 91-K(23)	42.82	3.59	1 714.07
84 (30)	193-V(6), 19-B(12), 101-L(10)	69.06	3.46	2 624.95
85 (30)	104-M(12), 100-L(15), 153-S(3)	343.55	6.73	12 427.81
86 (30)	95-K(30)	67.40	3.73	2 582.75
87 (30)	201-W(30)	841.55	11.59	30 149.38
88 (30)	111-M(30)	756.27	9.89	27 062.94
89 (30)	99-L(30)	647.70	9.19	23 220.91
90 (30)	124-O(30)	844.00	12.13	30 267.60

Table 12: Routes actually implemented at the SPAR Western Cape DC on 26 November 2019 (Continued).

Delivery (capacity)	**Route** ..., **Customer(load)**, ...	**Distance (kilometres)**	**Time (hours)**	**Cost (Rands)**
91 (30)	150-R(30)	315.20	5.55	11 365.19
92 (30)	109-M(30)	370.72	6.23	13 349.07
93 (30)	104-M(30)	141.82	4.45	5 230.39
94 (30)	15-B(30)	265.88	4.35	9 566.70
95 (30)	98-L(30)	319.10	5.10	11 474.50
96 (30)	99-L(10), 26-C(20)	743.66	12.56	26 781.34
97 (30)	149-R(30)	846.87	11.21	30 313.26
98 (30)	150-R(19), 149-R(7), 165-S(4)	939.73	20.23	34 104.23
99 (30)	110-M(7), 176-T(8), 174-T(14), 60-G(1)	310.58	6.24	11 244.59
100 (30)	172-S(20), 111-M(10)	758.06	9.29	27 089.31
101 (30)	109-M(28)	370.72	6.09	13 340.83
102 (30)	156-S(20), 132-P(10)	94.12	4.30	3 552.48
103 (30)	10-B(30)	371.25	6.86	13 405.14
104 (30)	76-H(20), 10-B(10)	371.24	6.45	13 380.43
105 (30)	17-B(29), 164-S(1)	99.2	2.33	3 611.80
106 (30)	53-E(14), 130-P(16)	61.82	2.86	2 335.03
107 (30)	3-A(26), 51-D(4)	69.46	3.10	2 617.13
108 (30)	29-C(28)	40.69	2.04	1 546.61
109 (30)	154-S(30)	49.64	2.82	1 906.65
110 (30)	11-B(2), 208-Z(11), 97-K(14)	76.70	3.24	2 879.10
111 (30)	69-G(30)	19.13	2.72	832.80
112 (30)	157-S(1), 193-V(6), 143-P(12), 121-O(11)	78.77	4.05	2 999.72
113 (30)	178-T(28), 61-G(2)	174.47	4.10	6 352.28
114 (30)	45-D(7), 175-T(23)	367.98	7.10	13 304.91
115 (30)	185-V(17)	186.7	3.98	6 773.02
116 (30)	19-B(2), 28-C(17), 101-L(11)	70.76	3.55	2 689.31
117 (30)	69-G(30)	19.13	2.72	832.80
118 (30)	147-R(7)	1 662.51	25.24	59 702.06
119 (30)	85-K(30)	1 668.16	27.56	60 038.99
120 (30)	20-B(30)	1 588.94	26.59	57 208.53
121 (30)	84-K(30)	1 713.54	26.92	61 589.38
122 (30)	116-N(5), 85-K(23)	1 670.46	27.02	60 087.25
123 (30)	20-B(18)	1 588.94	25.49	57 142.53
124 (30)	92-K(30)	196.46	3.91	7 110.72
125 (40)	22-B(36)	350.85	6.27	14 410.26
126 (40)	41-C(39)	462.72	8.07	18 992.95
127 (40)	23-B(40)	224.41	6.02	9 337.54
128 (40)	85-K(30)	1 668.16	27.56	68 379.78

Table 13: Routes actually implemented at the SPAR Western Cape DC on 26 November 2019 (Continued).