Journal of the Operational Research Society (2005) 56, 954–961

www.palgrave-journals.com/jors

A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem

İK Altınel[1]*, T Öncan[2]

[1]*Boğaziçi University, Bebek, İstanbul, Türkiye; and* [2]*Galatasaray University, Ortaköy, İstanbul, Türkiye*

In this work we are concerned with the Clarke–Wright savings method for the classical capacitated vehicle routing problem. This is an NP-hard problem and numerous heuristic solution methods have been proposed. They can be classified as the classical ones and metaheuristics. Recent developments have shown that classical heuristics do not compare with the best metaheuristic implementations. However, some of them are very fast and simple to implement. This explains the popularity of the Clarke–Wright savings method in practice and the motivation behind its enhancements. We follow this line of research and propose a new enhancement which differs from the previous ones in its saving criterion: Customer demands are considered in addition to distances. Based on the extensive computational experiments we can say that the new method is not only very fast but also very accurate.
Journal of the Operational Research Society (2005) **56,** 954–961. doi:10.1057/palgrave.jors.2601916
Published online 22 December 2004

Keywords: heuristics; vehicle routing problem

Introduction

The classical capacitated vehicle routing problems (CVRP) can be defined as designing a set of routes with minimum total routing cost for a fleet of m identical vehicles with capacity D which have to serve n customers with non-negative demand d_i from a depot subject to the following constraints:

(a) Each customer is served exactly once by exactly one vehicle.
(b) Each route starts and ends at the depot.
(c) The total demand on each route does not exceed vehicle capacity D.

In many instances, additional requirements and assumptions are added to this basic definition in order to model various aspects of real-life distribution problems. For a complete survey see the recent monograph edited by Toth and Vigo.[1] The CVRP is a difficult combinational optimization problem and known to be NP-hard.[2] The largest problem which has been solved to optimality contains 50 customers.[3] As a result of the inadequacy of exact methods, heuristics are widely used in practice. CVRP heuristics can be classified as classical heuristics and metaheuristics. Classical heuristics are unsophisticated and perform a limited exploration of the search space compared with metaheuristics. However, they are simple, which makes them easy to understand and easy to implement, and produce fairly good solutions very fast. Some of them are flexible and can be extended easily to handle many variants of the CVRP. For complete studies on classical CVRP heuristics see, for example, the recent works by Laporte *et al*[4] and Laporte and Semet.[5] Metaheuristics intensively use neighbourhood search methods to explore solution space without necessarily improving the objective function and sometimes allowing infeasible moves. The performance of the metaheuristics is usually much higher than the ones of classical heuristics, but they require much more computational effort to have their parameters finely tuned. The larger the set of parameters they have can increase their flexibility but also make them more context dependent and difficult to extend to other instances. These methods have been successfully applied to the CVRP, described in Laporte *et al*[4] and Gendreau *et al*.[6]

In their very recent work Cordeau *et al*[7] introduce *accuracy*, *speed*, *simplicity* and *flexibility* as what they believe as the most important attributes of a good heuristic and compare well-known classical heuristics and best available metaheuristics according to these four criteria. They report that none of the classical heuristics is as accurate and flexible as any one of the metaheuristics: but Clarke and Wright's savings heuristic (CW) is very fast and simple to implement, which probably explains its popularity. As a consequence, how one can improve the accuracy of CW without harming its speed and simplicity very much becomes an interesting question. A quick intuitive answer could be the consideration of additional information in its saving criterion.

*Correspondence: *İK Altınel, Department of Industrial Engineering, Boğaziçi University, Bebek, İstanbul 34342, Türkiye.*
E-mail: altinel@boun.edu.tr

All of the savings methods proposed for the CVRP use only distance information. However, the work by Vigo on the asymmetric capacitated vehicle routing problem (ACVRP),[8] and the works by Salhi and Nagy[9] and Wade and Salhi[10] on the single and multi-depot CVRP with backhauling have shown that the use of vehicle overloads, which is implicitly related to customer demands, in addition to distances in the traditional insertion costs can improve the solution quality. Motivated by their results we propose a new parallel savings heuristic which combines distances and customer demands in the saving criterion. This paper consists of five sections. We summarize CW and its enhancements in the next section in order to put our work into the right place. Section three is where we describe our new parallel savings heuristic. We report the results of our experimental study in the next section. The final section includes the conclusions.

Clarke–Wright savings heuristic and its enhancements

The CW algorithm is not only one of the earliest methods proposed for the solution of the CVRP, but also probably the most widely used one in commercial routing packages. Initially every customer is visited by a separate vehicle. This is not clearly a feasible solution since a fleet of n vehicles (one vehicle per customer) is required. Subsequently, routes are combined repeatedly by considering the *saving* in the routing cost, which is obtained by using one vehicle instead of two for the same set of customers. Then the saving obtained by merging routes $(0, \ldots, i, 0)$ and $(0, j, \ldots, 0)$ into the route $(0, \ldots, i, j, \ldots, 0)$ is

$$\begin{aligned} s_{ij} &= (c_{0i} + c_{i0} + c_{0j} + c_{j0}) - (c_{0i} + c_{ij} + c_{j0}) \\ &= c_{i0} + c_{0j} - c_{ij} \end{aligned} \tag{1}$$

At every iteration the feasible combination of two routes that leads to the largest saving in the routing cost is performed. The heuristic stops when a feasible merge of routes that leads to a saving is no longer possible. The *best feasible merge* version is also known as the parallel version of CW. In the sequential version the first saving s_{ki} or s_{jl} that can feasibly be used to extend the current route $(0, i, \ldots, j, 0)$ by merging with another route containing edge $(k, 0)$ or containing edge $(0, l)$, is determined first. Then the merge operation is implemented and repeated with the current route until no feasible merge is possible. In their recent work on the CVRP heuristics Laporte and Semet[5] report that the parallel version dominates the sequential version, based on their experiments on the symmetric instances of Christofides *et al.*[11]

Saving defined as in formula (1) becomes higher when the distance between customers i and j is smaller relative to their distances to the depot. As a consequence, the original CW tends to produce good routes at the beginning. Gaskell[12] and Yellow[13] addressed this weakness in their early works and proposed the following parameterized saving expression:

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij} \tag{2}$$

Here λ, which can only take positive values, is called *route shape* parameter. It prevents the formation of circumferenced routes which tend to be created by the original CW. Another way to improve the performance of the CW algorithm is to extend the parametric saving expression (2) to consider more information about the spatial distribution of the customers. One approach can be the use of asymmetry between customers i and j with respect to their distances to the depot. This was Paessens' motivation to introduce one new term and parameter to the saving expression (2):[14]

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij} + \mu |c_{0i} - c_{j0}| \tag{3}$$

Two other enhancements of CW are due to Golden *et al*[15] and Nelson *et al.*[16] They mainly contribute to its speed by using special data structures in order to make the determination of the maximum saving value computationally more efficient. The storage of the complete savings set allows the consideration of different selection strategies. One such example is Daskin's work.[17] At each iteration the original CW myopically chooses the route pairs. Taking into consideration this observation, Daskin has come up with the randomized savings approach. He proposes to run the standard CW but at the route merging step, the kth best saving is selected from the top of the saving list where k is a random variable between 1 and the *depth,* which is a parameter. The randomized algorithm is then repeated for a *number of times*, and the best solution is recorded.

A new heuristic for the CVRP

In this section, we introduce a new enhancement of the original CW. While running the original CW and its enhancements, especially towards the end, the merge of routes with equal or very close savings occur often. Then it may be more interesting to also consider customer demands while calculating savings. In fact, the VRP consists of two problems: The multiple travelling salesman problem (m–TSP) and the bin packing problem (BPP). Hence, a saving expression which somehow combines the solution strategies proposed for them can increase the chance of obtaining higher improvements. For this purpose we have adopted the well-known *first fit decrease* idea of Martello and Toth,[18] which was originally used for the BPP: *put first larger items.* In short we propose the following new saving criterion:

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij} + \mu |c_{0i} - c_{j0}| + \nu \frac{d_i + d_j}{\bar{d}} \tag{4}$$

Here d_i is the demand of customer i, $\bar{d}$ is the average demand used to normalize, c_{ij} is the distance between customers i and j, c_{i0} and c_{0j} are, respectively, distances between customers i

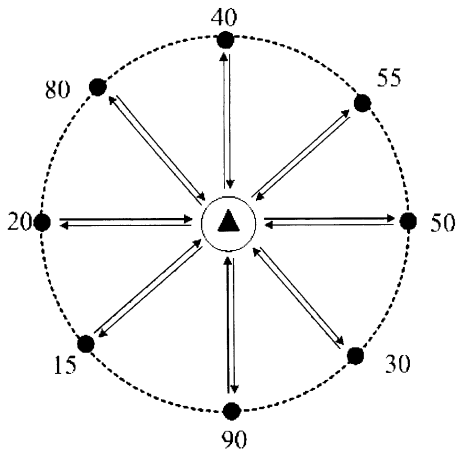

Figure 1 Effect of customer demand in merging.

and j, and the depot, and v is the new parameter. Since it also includes enhancements due to Gaskell,[12] Yellow[13] and Paessens,[14] saving expression (4) is more general. We consider demands normalized with the average demand $\bar{d} = (1/n)\sum_{i=1}^{n} d_i$ because we prefer to include the relative importance of the customers according to their demands on a vehicle's capacity. The new method is also a parallel heuristic and starts with n vehicles each visiting one customer. Then gradually merges routes $(0, i\ldots, 0)$ and $(0, \ldots, j, 0)$ into $(0, \ldots, i, j, \ldots, 0)$ if the saving criterion (4) has the largest value between all possible such route combinations.

The situation given in Figure 1 illustrates the effect of customer demands, namely the third term of the new saving expression (4). There are eight customer points located equidistantly from each others on a circle whose centre is the depot. Namely $c_{i,i+1} = c_{i+1,i+2}$ for $i = 1, \ldots, 6$ and $c_{0,i} = c_{0,i+1}$, for $i = 1, \ldots, 7$. The given numbers in this figure represent customer demands. Assume also that the fleet consists of identical vehicles with capacity 100 units. Observe that when customer demands are ignored, namely Paessens' saving expression (3) is used, any two adjacent customer points can be combined unless they violate capacity restriction. However, this is different for the situation when demands are also considered: for any positive v, customer points with demands 20 and 80 will have priority in our example.

Another strategy for introducing demand information into the saving criterion may be the consideration of the remaining vehicle capacities. Very briefly, one might be interested in the saving expression

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij} + \mu|c_{0i} - c_{j0}| + \gamma\frac{(D - d_i) + (D - d_j)}{D - \bar{d}} \quad (5)$$

Here the new parameter γ weights the relative importance given to customers with smaller demands, which results in higher remaining capacities. In other words the larger is γ, the smaller are the items put first, this time. As a result of extensive computational study we can say that its performance is not better than the one with the saving criterion (4). Hence we have preferred not to report the related computational results here.

Computational results

In order to demonstrate the improvement obtained by considering customer demands we conducted experiments with the original CW, its enhancements due to Gaskell,[12] Yellow[13] and Paessens,[14] and the new heuristic with savings criterion (4) on the same test bed. In our implementations we use Method 4 by Nelson *et al*,[16] which is efficient and not very complicated to implement. In this method, savings are stored in a heap represented as a $n \times (n+1)/2$ dimensional array. They also use an additional linear array of size $n \times (n+1)/2$ to link saving s_{ij} to customer pair (i, j) and *vice versa*. At each step, the root of the heap, which contains the largest saving value s_{ij}, is removed from the heap and the routes passing through customers i and j are merged. Saving values which are associated to customers on these two merged routes are also removed from the heap. Besides, if it is not possible to merge the routes due to the capacity restrictions, then the corresponding value is also removed from the heap. Nelson *et al*[16] propose four more methods to make CW more efficient. Methods 1–3 are less efficient than Method 4. Method 5 is more efficient, but unfortunately the used data structure requires certain maturity in computer programming. In effect, Method 4 is very efficient and simpler to programme. In an earlier version of this work[19] we used a matrix to store the saving values. This is the most simple data structure to implement and require only very basic programming skills. However, the search effort spent on the determination of largest saving value was prohibitive; we ended up with CPU times approximately 15 times larger than the ones reported here.

In sophisticated implementations of savings heuristics, a tour improvement step is repeated after every merge operation to decrease the length of the new route. This is usually realized by using local improvement heuristics suggested originally for the TSP. As for example Daskin uses Or-opt,[20] which is then followed by 2-opt after every merge and reports considerable improvement in the accuracy of his randomized savings heuristic.[17] This is also Laporte and Semet's motivation for using 3-opt with the best improvement strategy after every merge in their reputed parallel CW implementation.[5] For only the verification of our codes we compare our implementation of the CW with theirs on the seven symmetric instances of Christofides *et al*;[11] they are the ones with capacity restrictions. The results are summarized in Table 1. The entries of the first column are the instances. The letter C denotes that there is a restriction on the vehicle capacity. Following digits denote the number

of customers; the depot is not counted. Second column is adopted directly from Laporte *et al*'s work.[4] Note that we have two implementations. The third and fourth columns include final total route lengths computed by our plain and sophisticated implementations. Our sophisticated implementation has also local improvement steps. Once two routes are merged we first use Or-opt with strings of up to four vertices; sequences of four, three, two and one consecutive vertices (customers) are relocated in the new route so that it becomes shorter. The string whose relocation results in the best improvement (largest decrease of all possible relocations) is relocated. Then 4-opt[21] is utilized on the same route: Four edges are replaced with four others so that the length of the new route decreases. We again do the interchange which results in the best improvement. However, our plain implementation does not include any local improvement procedure.

As can be observed, our plain implementation finds larger values. This should be expected since it does not include any local improvement step. However, our sophisticated implementation results in equal or slightly lower values except instance *C100a*: The value our implementation computes is slightly higher. This is also expected since we spend more effort on local improvement than Laporte and Semet do. In short it seems we can rely on our CW implementation and use it for benchmarking. While proposing the new heuristic, our main goal is to increase the performance of the original CW as much as we can with minimal sacrifice in its simplicity and speed. As a result we prefer to use mainly our plain implementation in our computational experiments. We also believe this exposes the contribution introduced by demand information better.

Table 1 Total routing costs obtained by different CW implementations

Instance	*Laporte–Semet*	*Altınel–Öncan (Plain)*	*Altınel–Öncan*
C50	578.56	584.64	572.29
C75	888.04	907.39	888.04
C100a	878.70	886.83	880.62
C150	1128.24	1140.42	1128.24
C199	1386.84	1395.74	1370.11
C120	1048.53	1068.14	1046.03
C100b	824.42	833.51	824.42

In Tables 2–6 we report the relative deviations in percentage from the best-known values for different test sets. Our test bed does not consist only of the instances by Christofides *et al*;[11] these are used for Table 2 only. For Table 3 the instances are chosen from Christofides and Eilon's[22] test bed, while the test bed from Augerat *et al*[23] is being used for Tables 4–6. Best values are obtained from various resources. For Augerat *et al*'s, and Christofides and Eilon's test sets we use the best-known results reported at http://www.branchandcut.org (last access 5/2004). The best-known results for Christofides *et al*'s instances are obtained from the work of Rochat and Taillard.[24]

In all these tables the first column represents the instances. The second column includes the best-known values. The third and fourth columns are obtained by using our plain CW implementation. The fifth and sixth columns include values calculated by our plain implementation of Gaskell–Yellow CW enhancement (GY). The numbers in each cell are respectively route-shape parameter λ, and corresponding relative deviations. The entries of the seventh column are the total CPU times in seconds. This format is repeated in columns 8 and 9 for our plain implementation of Paessens' enhancement (*P*) and in columns 11–13 for our plain implementation of the new heuristic (NEW) with (λ, μ) pairs and (λ, μ, ν) triplets, respectively. The search effort becomes higher with the increase in the number of parameters. In all the experiments λ changes within [0.1, 2], and μ and ν change within [0, 2], all with an increment of 0.1. In other words, the numbers in columns 5 and 6 are the best and the numbers in column 7 are the total of 20 values each of which is obtained by one run of GY. Similarly the numbers in columns 8 and 9 are the best and the numbers in column 10 are the total of $20 \times 21 = 420$ values. The search effort for NEW is even higher because of the third parameter ν. As a result the numbers in columns 11–13 are, respectively, the best and the

Table 2 Relative deviations for Christofides *et al*'s[11] test set

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
C50	524.61	11.442	0.02	1.3	11.217	0.06	1.0;0.3	8.425	1.17	1.4;0.9;0.3	5.898	24.93
C75	835.26	8.636	0.02	1.1	3.719	0.13	1.0;0.1	3.716	3.17	1.2;0.2;0.8	2.987	63.59
C100	826.14	7.346	0.02	1.3	6.443	0.22	1.3;0.8	4.951	5.50	1.4;0.3;0.3	4.265	117.94
C150	1028.42	10.890	0.03	1.1	8.484	0.63	1.6;0.4	6.913	13.73	1.6;0.3;0.4	6.248	283.03
C199	1291.45	8.075	0.06	1.1	6.952	1.34	1.4;0.2	6.086	28.59	1.3;0.0;1.1	5.291	627.70
C120	1042.11	2.498	0.02	1.0	2.498	0.41	1.3;0.3	2.340	8.45	1.1;0.1;0.3	1.506	182.96
C100b	819.56	1.702	0.02	1.1	1.304	0.27	1.3;0.3	0.972	5.63	1.3;0.3;0.6	0.629	119.72
Average		7.227	0.02		5.802	0.44		4.772	9.46		3.832	202.84

Table 3 Relative deviations on Christofides and Eilon's[22] test set

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
E-n22-k4	375	3.673	0.00	0.8	3.345	0.00	1.5;0.6	0.075	0.22	1.0;0.9;1.6	0.075	3.43
E-n23-k3	569	9.154	0.00	1.6	2.778	0.00	1.8;0.5	0.706	0.22	1.7;0.5;0.9	0.706	4.94
E-n30-k4	503	6.252	0.00	1.6	1.968	0.02	1.3;0.3	0.730	0.38	1.3;0.3;0.0	0.730	7.14
E-n33-k4	835	0.970	0.00	1.0	0.970	0.02	1.0;0.0	0.970	0.47	1.0;0.0;0.0	0.970	9.14
E-n76-k14	1021	3.291	0.00	1.1	3.284	0.16	1.1;0.1	3.069	3.30	1.3;0.0;1.0	2.355	64.17
E-n76-k8	735	8.128	0.02	1.4	7.013	0.16	1.7;1.1	6.339	3.13	1.6;0.3;0.7	5.463	62.91
E-n76-k7	682	8.230	0.00	1.5	6.335	0.14	1.6;0.8	5.232	3.12	1.7;0.8;0.1	4.349	62.53
E-n101-k14	1071	6.356	0.02	1.8	5.993	0.28	0.7;0.5	5.882	5.74	0.8;0.6;0.3	5.172	119.40
Average		5.757	0.00		3.961	0.10		2.875	2.07		2.477	41.71

Table 4 Relative deviations on Augerat *et al*'s[23] test set A

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
A-n32-k5	784	7.613	0.00	1.2	6.792	0.02	0.8;0.6	5.702	0.39	0.8;0.6;0.0	5.702	8.50
A-n33-k5	661	7.723	0.00	1.1	4.590	0.03	1.2;0.8	2.833	0.45	1.2;0.8;0.0	2.833	9.69
A-n33-k6	742	4.617	0.00	1.2	1.917	0.02	1.9;1.1	0.649	0.47	1.2;0.0;1.0	0.163	9.33
A-n34-k5	778	1.935	0.02	0.8	1.935	0.03	0.7;0.1	1.935	0.45	0.6;0.3;1.2	1.935	10.01
A-n36-k5	799	3.689	0.00	0.9	0.974	0.02	0.9;0.0	0.974	0.66	0.8;0.0;0.1	0.974	10.82
A-n37-k5	669	5.801	0.02	1.3	5.430	0.03	0.9;0.9	4.381	0.62	1.5;0.3;1.1	3.802	11.80
A-n37-k6	949	2.909	0.02	1.0	2.909	0.02	1.0;0.1	2.847	0.64	1.1;0.1;0.3	2.694	12.04
A-n38-k5	730	5.224	0.00	1.0	5.224	0.05	1.6;0.1	4.125	0.64	1.4;0.3;0.2	3.577	12.51
A-n39-k5	822	9.731	0.02	1.4	3.559	0.05	1.1;0.1	3.193	0.72	1.1;0.1;0.0	3.193	13.35
A-n39-k6	831	3.861	0.00	0.9	3.350	0.05	0.8;0.2	2.233	0.64	0.8;0.2;0.0	2.233	13.22
A-n44-k7	937	4.166	0.00	0.9	4.166	0.05	1.9;0.8	3.399	0.92	1.6;0.4;1.8	2.394	18.21
A-n45-k6	944	6.615	0.02	1.1	3.772	0.05	1.2;0.2	2.305	0.94	1.0;0.0;1.4	1.383	18.37
A-n45-k7	1146	4.710	0.00	0.9	4.710	0.03	2.0;1.0	2.007	1.00	1.8;0.2;1.8	1.842	18.97
A-n46-k7	914	2.817	0.00	1.3	2.289	0.05	1.1;0.1	2.152	1.00	1.1;0.1;0.0	2.152	19.86
A-n48-k7	1073	3.711	0.02	1.2	3.018	0.05	1.6;0.7	2.911	1.13	1.6;0.7;0.0	2.911	22.11
A-n53-k7	1010	8.856	0.02	0.7	7.919	0.06	1.5;0.6	3.563	1.34	1.9;1.2;1.6	3.427	27.56
A-n54-k7	1167	2.931	0.00	1.0	2.931	0.06	1.7;0.9	2.106	1.44	1.1;0.1;0.9	0.861	28.62
A-n55-k9	1073	2.501	0.00	1.0	2.501	0.08	1.3;0.2	2.475	1.59	1.1;0.1;1.0	2.378	30.56
A-n60-k9	1354	5.013	0.00	1.1	3.654	0.09	1.9;0.8	2.326	1.84	1.4;0.0;0.9	1.640	37.88
A-n61-k9	1034	6.599	0.00	1.1	1.680	0.08	1.1;0.0	1.680	1.94	1.1;0.0;0.1	1.654	38.39
A-n62-k8	1288	5.032	0.00	1.0	5.032	0.09	1.2;0.2	4.901	1.98	1.0;0.0;0.2	4.648	40.50
A-n63-k10	1314	3.425	0.00	1.1	2.068	0.08	1.3;0.2	2.066	2.09	1.4;0.4;1.3	1.941	41.25
A-n64-k9	1401	6.133	0.00	1.4	4.396	0.11	1.4;0.5	2.986	2.15	1.4;0.6;0.1	2.497	42.53
A-n63-k9	1616	4.453	0.00	0.8	4.261	0.09	1.6;0.6	2.078	2.09	1.6;0.6;0.1	2.051	41.57
A-n65-k9	1174	5.572	0.02	0.9	4.777	0.09	0.9;0.3	4.416	2.12	0.9;0.1;0.3	2.392	43.26
A-n69-k9	1159	4.468	0.02	1.3	2.250	0.11	1.3;0.0	2.250	2.53	1.3;0.0;0.0	2.250	50.45
A-n80-k10	1763	5.555	0.02	0.8	3.818	0.17	1.8;0.6	3.261	3.53	1.5;0.7;1.6	2.952	71.03
Average		5.024	0.01		3.701	0.06		2.806	1.31		2.462	26.01

total of $20 \times 21 \times 21 = 8820$ values. As for example, for instance C50, it takes NEW 25 s, which makes 0.003 s per run in the average, to obtain a total cost of 555.55 and parameters $(\lambda, \mu, \nu) = (1.4;0.9;0.3)$ as the best of 8820 values. The values given in the last rows are columnwise averages of the relative percent deviations and average of total CPU times. For example, they go from 7.966% (for the CW) down to 2.736% (for NEW) in Table 6, which means approximately 5.23% relative improvement in the average accuracy. Based on these values one can easily agree that the consideration of customer demands increase considerably the accuracy of CW with a reasonable decrease in the efficiency. Negative entries, although they are very few, represent values lower than the best knowns.

Table 5 Relative deviations on Augerat *et al*'s[23] test set B

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
B-n31-k5	672	1.364	0.00	0.9	1.107	0.02	1.9;1.4	1.045	0.40	0.9;0.0;0.1	0.795	7.70
B-n34-k5	788	0.804	0.00	1.2	0.235	0.03	1.2;0.0	0.235	0.51	1.2;0.0;0.0	0.235	9.80
B-n35-k5	955	2.443	0.00	1.0	2.443	0.03	1.5;1.2	2.256	0.52	1.1;0.1;1.8	2.145	10.55
B-n38-k6	805	3.365	0.00	0.9	3.150	0.03	1.4;0.4	2.360	0.69	1.5;0.5;0.2	2.283	12.54
B-n39-k5	549	3.225	0.00	1.1	1.118	0.05	1.4;0.3	1.092	0.70	1.4;0.3;0.0	1.092	13.41
B-n41-k6	829	8.334	0.00	0.8	6.614	0.05	0.7;0.6	5.940	0.73	0.8;0.7;0.2	3.941	15.08
B-n43-k6	742	5.386	0.00	0.8	2.151	0.03	1.2;0.4	1.741	0.80	0.9;0.1;0.4	1.741	16.62
B-n44-k7	909	3.161	0.02	1.0	3.161	0.05	1.8;0.8	2.859	0.94	1.9;0.9;1.8	2.825	18.07
B-n45-k5	751	0.821	0.02	1.0	0.821	0.03	1.0;0.0	0.821	0.84	1.1;0.0;0.8	0.494	18.27
B-n45-k6	678	7.351	0.00	1.1	6.262	0.05	0.9;0.6	5.198	0.89	0.9;0.7;0.8	5.119	18.65
B-n50-k7	741	1.053	0.02	1.1	0.935	0.06	1.1;0.1	0.590	1.22	1.0;0.0;0.2	0.590	23.40
B-n50-k8	1312	3.204	0.02	0.9	3.065	0.06	1.5;0.6	2.461	1.23	1.9;1.0;0.3	2.008	24.92
B-n51-k7	1032	8.648	0.02	1.2	−0.326	0.06	1.2;0.1	−0.424	1.22	1.2;0.0;0.8	−0.597	23.83
B-n52-k7	747	2.396	0.00	1.0	2.396	0.05	1.0;0.0	2.396	1.34	1.3;0.0;1.5	1.326	25.73
B-n56-k7	707	3.782	0.00	0.7	2.372	0.08	1.3;0.5	2.279	1.53	0.8;0.0;0.2	2.209	30.71
B-n57-k7	1153	7.526	0.00	1.3	1.249	0.06	1.7;0.7	0.330	1.72	1.1;0.0;0.5	−0.349	32.39
B-n57-k9	1598	3.468	0.00	0.9	1.359	0.08	0.9;0.0	1.359	1.64	0.9;0.0;0.0	1.359	32.92
B-n63-k10	1496	6.830	0.00	0.9	4.452	0.11	1.8;0.8	4.014	2.08	1.8;0.8;0.0	4.014	41.85
B-n64-k9	861	7.033	0.00	1.0	7.033	0.11	1.8;0.9	6.759	2.08	1.1;0.7;1.8	5.815	41.92
B-n66-k9	1316	7.631	0.02	0.8	5.530	0.11	1.9;1.2	2.900	2.30	1.7;0.9;0.9	2.668	45.31
B-n67-k10	1032	6.584	0.02	1.2	6.457	0.13	1.3;0.5	3.828	2.28	1.2;0.4;0.3	3.015	46.82
B-n68-k9	1272	3.598	0.02	1.0	3.598	0.11	1.0;0.0	3.598	2.39	1.0;0.0;0.2	3.470	49.17
B-n78-k10	1221	3.568	0.02	1.0	3.568	0.17	1.3;0.4	3.456	3.36	1.0;0.1;0.9	3.305	67.77
Average		4.416	0.01		2.989	0.07		2.482	1.37		2.152	27.28

Table 6 Relative deviations on Augerat *et al*'s[23] test set *P*

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
P-n16-k8	450	6.394	0.00	1.3	5.285	0.00	1.9;0.5	0.433	0.14	1.8;0.3;1.7	0.433	2.07
P-n19-k2	212	12.214	0.00	1.3	4.077	0.02	0.6;0.7	4.077	0.16	0.6;0.7;0.0	4.077	2.76
P-n20-k2	216	8.331	0.00	0.4	8.331	0.02	1.8;1.2	6.222	0.14	1.5;1.1;1.6	6.222	3.02
P-n21-k2	211	11.936	0.00	0.4	11.936	0.02	1.8;1.2	9.738	0.19	1.3;1.0;2.0	9.054	3.05
P-n22-k2	216	10.879	0.00	0.4	10.879	0.00	1.9;0.7	4.366	0.22	1.9;0.7;0.8	0.866	3.33
P-n22-k8	603	−2.053	0.00	0.9	−2.257	0.00	0.9;0.0	−2.257	0.19	0.9;0.0;0.0	−2.257	4.11
P-n23-k8	529	1.980	0.00	1.2	1.458	0.15	1.5;0.2	1.458	0.22	1.4;0.2;1.3	1.458	4.60
P-n40-k5	458	13.182	0.00	1.3	7.932	0.05	1.0;1.0	2.228	0.70	0.9;1.0;0.4	2.228	13.67
P-n45-k5	510	12.344	0.00	1.3	4.164	0.05	1.9;0.7	2.728	0.92	1.5;0.1;0.7	2.434	17.92
P-n50-k10	696	6.299	0.00	1.5	3.986	0.06	1.2;0.1	2.409	1.19	1.2;0.1;0.0	2.409	24.08
P-n50-k7	554	7.767	0.00	1.7	6.073	0.05	1.8;0.5	4.577	1.17	1.7;0.5;1.6	4.284	23.35
P-n50-k8	631	6.868	0.00	1.5	3.060	0.06	1.4;0.2	2.464	1.22	1.2;0.2;0.7	2.464	23.83
P-n51-k10	741	6.744	0.00	1.7	4.611	0.08	1.5;0.6	2.147	1.28	1.0;0.3;0.1	1.887	25.00
P-n55-k10	694	6.116	0.02	1.9	4.197	0.08	1.4;0.3	3.179	1.50	1.2;0.1;1.7	3.056	30.67
P-n55-k15	989	−1.105	0.02	1.0	−1.105	0.08	1.6;0.9	−2.596	1.60	1.9;0.8;0.7	−2.783	32.25
P-n55-k7	568	8.923	0.00	1.2	3.794	0.06	1.4;0.4	3.200	1.51	2.0;0.8;1.7	2.990	29.33
P-n55-k8	576	9.665	0.02	1.4	3.955	0.08	1.3;0.3	3.210	1.55	1.3;0.3;1.9	2.091	28.84
P-n60-k10	744	7.553	0.00	0.9	7.140	0.09	1.9;0.6	4.737	1.84	2.0;0.5;0.3	2.739	36.16
P-n60-k15	968	5.058	0.00	0.8	4.023	0.09	0.8;0.0	4.023	1.86	0.9;0.0;0.5	3.593	37.97
P-n70-k10	834	7.538	0.00	0.8	4.465	0.13	0.6;0.4	2.391	2.58	0.6;0.4;;0.0	2.391	52.26
P-n76-k4	593	15.495	0.01	1.1	10.289	0.16	1.7;0.8	6.790	2.91	1.7;0.8;0.0	6.790	61.74
P-n76-k5	627	13.138	0.02	2.0	8.095	0.16	1.6;0.8	5.145	3.06	2.0;0.5;1.5	3.768	61.79
Average		7.966	0.00		5.199	0.07		3.212	1.19		2.736	23.72

Table 7 The effect local improvement on Christofides and Eilon's[22] test set

		CW		*GY*			*P*			*NEW*		
Instance	*Best*	*% Dev*	*CPU*	λ	*%Dev*	*CPU*	(λ, μ)	*% Dev*	*CPU*	(λ, μ, ν)	*% Dev*	*CPU*
E-n22-k4	375	3.672	0.01	0.8	3.345	0.02	1.4;0.6	0.075	0.31	1.2;0.2;1.1	0.075	6.62
E-n23-k3	569	−0.077	0.00	0.8	−0.077	0.14	0.8;0.0	−0.077	2.56	0.8;0.0;0.0	−0.077	52.83
E-n30-k4	503	1.022	0.00	1.6	0.754	0.17	1.9;1.0	0.599	3.23	1.0;0.0;1.1	0.400	60.25
E-n33-k4	835	0.934	0.00	1.0	0.934	0.14	0.9;1.7	0.770	2.65	0.9;1.4;1.1	0.650	54.33
E-n76-k14	1021	3.211	0.00	1.1	3.195	0.16	1.1;0.1	3.074	3.02	1.3;0.0;0.4	2.355	65.71
E-n76-k8	735	6.888	0.02	1.1	6.683	0.39	0.9;0.5	5.146	7.34	1.3;0.1;1.3	3.668	145.38
E-n76-k7	682	6.534	0.02	0.7	6.534	0.31	0.7;1.3	3.425	5.94	0.7;1.3;0.0	3.425	121.57
E-n101-k14	1071	6.143	0.02	1.8	4.861	0.38	1.6;1.2	4.683	7.08	0.7;0.6;1.3	4.210	145.53
Average		3.541	0.01		3.279	0.21		2.212	4.02		1.838	81.53

One immediate question at this point is probably how much additional improvement sophisticated implementation can provide. We have prepared Table 7 on Christofides and Eilon's test set in order to give an idea about the answer. Parameters λ, μ and ν are selected as before. The values reported for CW, GY, *P* and NEW are all computed with the same local improvement steps, namely first Or-opt, then 4-opt this time. At the first look at the last rows of both Tables 3 and 7 we can say that local improvement steps have resulted in slight improvements on the relative deviations (25.79% in the average for NEW) with a considerable increase in the CPU times (95.4% in the average for NEW).

The reported values of the parameters inform us implicitly about the contribution of their corresponding term to the savings, for example, when $\lambda = 1$ GY becomes CW. Similarly for $\lambda = 1$, $\mu = \nu = 0$ NEW's behaviour is equivalent to CW's. Therefore, whether the consideration of customer demand is a good idea or not can be determined from how often ν is not zero. For 69 out of 87 values reported in column 11 of Tables 2 and 6 (79% of the reported best values) ν is different than zero, which also points the contribution of NEW.

Finally, we should point out that the computational tests are realized on a Sun Microsystems Blade V–V1000 with a 750 MHz Ultrasparc III CPU and 2 Gbyte RAM, working within SOLARIS 8 environment. All of our codes are written in C^{++} language.

Conclusion

As is reported in two recent works[5,7] metaheuristics perform better than classical heuristics for solving the capacitated vehicle routing problem. However, Clarke–Wright savings heuristic is very simple to implement and also very fast. These are probably the reasons of its popularity and explain its wide usage in much commercial software. Thus any enhancement which improves its accuracy without harming its simplicity and speed would be a contribution with wide applicability. In this work a new enhancement of the CW savings heuristic for the CVRP has been introduced. Compared with the previous enhancements this new one also considers customer demands in its saving criterion. This is realized by introducing a new term and multiplier ν. Although there is an increase in the search effort because of the new parameter, the improvement in the accuracy of the original CW savings heuristic is remarkable. We observe 3.395, 3.279, 2.562, 2.264 and 5.230% relative improvements in the average relative deviations obtained, respectively, on Christofides *et al*'s,[11] Christofies and Eilon's,[22] and Augerat *et al*'s A, B and P test sets.[23] These improvements can be made even higher by introducing local improvement steps after every merge operation. Unfortunately this does not seem worthwhile when the increase in the accuracy is compared with the decrease in the efficiency.

Acknowledgements—This research has been partially supported by Boğaziçi University Research Fund Grant No: 04HA301D.

References

1 Toth P and Vigo D (eds) (2001). *The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications*. SIAM publishing: Philadelphia, PA.
2 Lenstra J and Rinnooy Kan A (1981). Complexity of vehicle routing and scheduling problems. *Networks* **11**: 221–227.
3 Toth P and Vigo D (1998). Exact solution of the vehicle routing problem. In: Crainic TG and Laporte G (eds). *Fleet Management and Logistics*. Kluwer, Boston, pp 1–31.
4 Laporte G, Gendreau M, Potvin J and Semet F (2000). Classical and modern heuristics for the vehicle routing problem. *Int Trans Opns Res* **7**: 285–300.
5 Laporte G and Semet F (2001). Classical heuristics for the vehicle routing problem. In: Toth P and Vigo D (eds). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM Publishing, Philadelphia, PA, pp 109–128.
6 Gendreau M, Laporte G and Potvin J (2001). Metaheuristics for the capacitated VRP. In: Toth P and Vigo D (eds). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics

and Applications. SIAM Publishing, Philadelphia, PA, pp 129–154.
7 Cordeau J-F *et al* (2002). A guide to vehicle routing heuristics. *J Opl Res Soc* **53**: 512–522.
8 Vigo D (1996). A heuristic algorithm for the asymmetric capacitated vehicle routing problem. *Eur J Opl Res* **89**: 108–126.
9 Salhi S and Nagy G (1999). A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *J Opl Res Soc* **50**: 1034–1042.
10 Wade AC and Salhi S (2002). An investigation into a new class of vehicle routing problem with backhauls. *Omega* **30**: 479–487.
11 Christofides N, Mingozzi A and Toth P (1979). The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P and Sandi C (eds). *Combinatorial Optimization*. Wiley, Chichester, pp 315–338.
12 Gaskell TJ (1967). Bases for vehicle fleet scheduling. *Opl Res Quart* **18**: 281–295.
13 Yellow P (1970). A computational modification to the savings method of vehicle scheduling. *Opl Res Quart* **21**: 281–283.
14 Paessens H (1988). The savings algorithm for the vehicle routing problem. *Eur J Opl Res* **34**: 336–344.
15 Golden BL, Magnanti TL and Nguyen HQ (1977). Implementing vehicle routing algorithms. *Networks* **7**: 340–349.
16 Nelson MD, Nygard KE, Griffin JH and Shreve WE (1988). Implementation techniques for the vehicle routing problem. *Comp Opns Res* **12**: 273–283.
17 Daskin MS (2002). Private communication.
18 Martello S and Toth P (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons: New York.
19 Altınel İK and Öncan T (2003). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem, FBE-IE-02/2003-02. Institute for Graduate Studies in Science and Engineering, Boğaziçi University, Bebek, İstanbul, Türkiye.
20 Or İ (1976). *Traveling salesman–type combinational problems and their relation to the logistics of regional blood banking*. PhD dissertation, Northwestern University, Evanston, IL.
21 Lin S (1965). Computer solutions of the traveling salesman problem. *Bell System Comput J* **44**: 2245–2269.
22 Christofides N and Eilon S (1969). An algorithm for the vehicle dispatching problem. *Opl Res Qhart* **20**: 309–318.
23 Augerat P *et al* (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. *Technical Report RR 949-M* Université Joseph Fourier, Grenoble.
24 Rochat Y and Taillard ED (1995). Probabilistic diversification and intensification in local search for vehicle routing. *J Heuristics* **1**: 147–167.

Received April 2003;
accepted September 2004 after three revisions