Journal of the Operational Research Society (2011) 62, 1085–1097

www.palgrave-journals.com/jors/

On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics

AA Juan[1], J Faulin[2]*, J Jorba[1], D Riera[1], D Masip[1] and B Barrios[2]

[1]*Open University of Catalonia, Barcelona, Spain; and* [2]*Public University of Navarre, Pamplona, Spain*

This paper presents the SR-GCWS-CS probabilistic algorithm that combines Monte Carlo simulation with splitting techniques and the Clarke and Wright savings heuristic to find competitive quasi-optimal solutions to the Capacitated Vehicle Routing Problem (CVRP) in reasonable response times. The algorithm, which does not require complex fine-tuning processes, can be used as an alternative to other metaheuristics—such as Simulated Annealing, Tabu Search, Genetic Algorithms, Ant Colony Optimization or GRASP, which might be more difficult to implement and which might require non-trivial fine-tuning processes—when solving CVRP instances. As discussed in the paper, the probabilistic approach presented here aims to provide a relatively simple and yet flexible algorithm which benefits from: (a) the use of the geometric distribution to guide the random search process, and (b) efficient cache and splitting techniques that contribute to significantly reduce computational times. The algorithm is validated through a set of CVRP standard benchmarks and competitive results are obtained in all tested cases. Future work regarding the use of parallel programming to efficiently solve large-scale CVRP instances is discussed. Finally, it is important to notice that some of the principles of the approach presented here might serve as a base to develop similar algorithms for other routing and scheduling combinatorial problems.

Journal of the Operational Research Society (2011) **62,** 1085–1097. doi:10.1057/jors.2010.29
Published online 19 May 2010

Keywords: simulation; heuristics; probabilistic algorithms; vehicle routing; road transport

Introduction

Road transportation is the predominant way of transporting goods in many parts of the world. Direct costs associated with this type of transportation have experienced a significant increase since 2000, and more so during these last years because of the rise of oil prices. Furthermore, road transportation is intrinsically associated with indirect or external costs, which are easily observable—congestion, contamination, security-related and safety-related costs, mobility, delay-time costs, etc—but are usually left unaccounted for because it is not easy to quantify them (Kumares and Labi, 2007). For example, traffic jams in metropolitan areas constitute a serious challenge to the competitiveness of European industry. According to some studies (Bastiaans, 2000), external costs because of traffic jams represent about 2% of the European Gross Domestic Product (GDP). This percentage has followed an increasing trend during the last 5 years. In addition to these tangible costs we could consider many others, such as environmental costs because of the production and use of fossil fuel. Therefore, there is a need for developing efficient models and methods that support decision-making processes in the road transportation arena so that optimal (or quasi-optimal) strategies can be chosen. This necessity for optimizing road transportation, which affects both the public and private sectors, constitutes a major challenge for most industrialized regions.

This paper focuses on the so-called Capacitated Vehicle Routing Problem (CVRP), which is one of the best-known and simplest variants of the Vehicle Routing Problem (VRP). Consider a complete graph G = (N, E), where:

- $N = \{0, 1, \ldots, n\}$ is a set of nodes representing the central depot (node 0) and the n customers or clients to be supplied (nodes 1 to n),
- E = {(i, j)|i, j in N with i < j} are the edges (ie roads) connecting these nodes, and
- $C_{ij} = C_{ji}$ are the travelling cost from node i to node j for all i, j in N with i < j.

Assume also that a demand d_i is defined for each customer (i in N-{0}). All these demands must be served by a set of vehicles V = {1, …, k} with a given maximum capacity Q per vehicle. Under these circumstances, the main goal here is to find a minimal-cost feasible solution, that is, a set of k (or fewer) routes which minimizes the total distribution

*Correspondence: J Faulin, Public University of Navarre, Campus Arrosadia, 31006 Pamplona, Spain.
E-mail: javier.faulin@unavarra.es

costs while satisfying all customers' demands and vehicle capacity constraints.

The CVRP is an NP-hard problem, which implies a non-polynomial increase of the space of solutions' size when the number of nodes is increased. Although this problem has already been studied for decades, it is still attracting researchers' attention due to its potential applications both to real-life scenarios and to the development of new algorithms, optimization methods and metaheuristics for solving combinatorial problems (Golden *et al*, 2008). In fact, different approaches to the CVRP have been explored during the last decades. These approaches range from the use of pure optimization methods—such as linear programming to solve small-size to medium-size problems with relatively simple constraints—to the use of heuristics and metaheuristics—which provide quasi-optimal solutions for medium and large-size problems with more complex constraints (Cordeau *et al*, 2004).

Nevertheless, most of the methods previously cited focus on the minimization of an aprioristic cost function—which usually models tangible costs—subject to a set of well-defined constraints. However, real-life problems are much more complex. These include intangible costs, fuzzy constraints, and desirable solution properties that are difficult to model (Poot *et al*, 2002; Kant *et al*, 2008). In other words, it is not always straightforward to build a model that takes into account all possible costs (eg, environmental costs, work risk, etc), constraints and desirable solution properties (eg, time or geographical restrictions, balanced work load among routes, solution attractiveness, etc). For that reason, it becomes necessary to provide not only an optimal or quasi-optimal solution, but also a set of alternative quasi-optimal solutions with different properties, such that decision-makers can choose among them according to their specific needs and preferences, that is, according to their utility function, which is usually unknown to the researcher. Moreover, as some researchers have already pointed out, there is a need for more simple and flexible methods to solve the problem when considering the numerous side constraints that arise in practice (Laporte, 2007).

This paper introduces the SR-GCWS-CS probabilistic algorithm, which combines Monte Carlo simulation with the Clarke and Wright savings heuristic and with some splitting and cache (memory-based) techniques to efficiently provide a set of alternative quasi-optimal solutions for the CVRP.

Related work and original contributions

Clarke and Wright's Savings (CWS) constructive algorithm (Clarke and Wright, 1964) is probably the most cited heuristic to solve the CVRP. The CWS is an iterative method that starts out by considering an initial dummy solution in which each customer is served by a dedicated vehicle. Next, the algorithm initiates an iterative process for merging some of the routes in the initial solution. Merging routes can improve the expensive initial solution so that a unique vehicle serves the nodes of the merged route. The merging criterion is based upon the concept of *savings*. Roughly speaking, given a pair of nodes to be served, a savings value can be assigned to the edge connecting these two nodes. This savings value is given by the reduction in the total cost function due to serving both nodes with the same vehicle instead of using a dedicated vehicle to serve each node—as proposed in the initial dummy solution. This way, the algorithm constructs a list of savings, one for each possible edge connecting two demanding nodes. At each iteration of the merging process, the edge with the largest possible savings is selected from the list as far as the following conditions are satisfied: (a) the nodes defining the edge are adjacent to the depot, and (b) the two corresponding routes can be feasibly merged—that is, the vehicle capacity is not exceeded after the merging. The CWS algorithm usually provides relatively good solutions, especially for small- and medium-size problems, but it also presents difficulties in some cases (Gaskell, 1967). Many variants and improvements of the CWS have been proposed in the literature. For instance, Mole and Jameson (1976) generalized the definition of the savings function, introducing two parameters for controlling the savings behaviour. Similarly, Holmes and Parker (1976) developed a procedure based on the CWS algorithm, using the same savings function but introducing a solution perturbation scheme to avoid poor quality routes. Beasley (1981) adapted the CWS method to use it to optimize inter-customer travel times. Correspondingly, Dror and Trudeau (1986) developed a version of the CWS method for the Stochastic VRP. Two years later, Paessens (1988) analyzed the main characteristics of the CWS method and its performance in generic VRP. Recently, the CWS heuristic has been finely tuned by means of Genetic Algorithm experimentation by Battarra *et al* (2009). For a more comprehensive discussion on the various CWS variants, the reader is referred to Toth and Vigo (2002) and Laporte (2007).

By using constructive heuristics as a basis, metaheuristics became popular for the VRP during the nineties. Some early examples are the Tabu Route method by Gendreau *et al* (1994) or the Boneroute method of Tarantilis and Kiranoudis (2002). Tabu Search algorithms, like those proposed by Taillard (1993) or Toth and Vigo (2003) are among the most cited metaheuristics. Genetic Algorithms have also played a major role in the development of effective approaches for the VRP. Some examples are the studies of Berger and Barkaoui (2003), Prins (2004), Mester and Bräysy (2007) or Nagata (2007). Another important approach to the VRP is given by the Greedy Randomized Adaptive Search Procedure or GRASP (Feo and Resende, 1995; Resende, 2008; Festa and Resende, 2009a). A GRASP algorithm is a multi-start or iterative process in which each GRASP iteration consists of two phases: a construction phase—in which a feasible solution is produced—and a local-search phase—in which a local optimum in the neighbourhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements in a candidate list according to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by the random choice of one of the best candidates in the list, but not necessarily the top candidate. This choice technique allows for different solutions to be obtained at each GRASP iteration. As the algorithm proposed in this paper is a probabilistic one and it makes use of a savings (candidates) list, the logic behind our approach is related to the logic behind GRASP algorithms—this issue will be further discussed later. Traditionally, however, GRASP techniques have not focused on the CVRP arena but, instead, they have focused on other VRP variants, that is, VRP with time windows (VRPTW), production-distribution problems or location-routing problems. For these VRP variants, several hybrid GRASP algorithms have been recently proposed (Gouvêa *et al*, 2007; Boudia *et al*, 2007; Duhamel *et al*, 2010).

As stated in the Introduction section, the SR-GCWS-CS algorithm is based on the combination of the CWS heuristic with Monte Carlo simulation (MCS). Additionally, the algorithm incorporates some memory capabilities—which are provided by a hash table of best-known routes—and specific splitting techniques which are based on the geometrical properties of intermediate solutions. The joint use of memory capabilities and splitting techniques contributes to a significant improvement in the overall performance of the hybrid CWS-MCS base algorithm. MCS can be defined as a set of techniques that make use of random numbers and statistical distributions to solve certain stochastic and deterministic problems (Law, 2007). MCS has proved to be extremely useful for obtaining numerical solutions to complex problems that cannot be efficiently solved by using analytical approaches. Buxey (1979) was probably the first author to combine MCS with the CWS algorithm to develop a procedure for the CVRP. This method was revisited by Faulin and Juan (2008), who introduced an entropy function to guide the random selection of nodes. MCS has also been used by Fernández de Córdoba *et al* (2000), Juan *et al* (2008), Faulin *et al* (2008) and Juan *et al* (2009) to solve the CVRP. However, despite the interesting contributions and ideas provided in those initial works, the results obtained were not competitive enough with state-of-the-art metaheuristics for the CVRP. On the contrary, the algorithm presented in this paper overcomes these limitations and accomplishes its goals efficiently, providing state-of-the-art solutions for all tested benchmarks in reasonable times. Therefore, as the main contribution of this paper, we present a hybrid approach that has proved to be an efficient procedure for obtaining quasi-optimal solutions in small- and medium-size CVRP instances and, at the same time, offers some additional advantages over other existing metaheuristics, namely: (a) it is relatively simple to implement, which facilitates its use in practical applications; (b) it is a robust and flexible methodology that can be easily adapted to consider additional constraints and costs; (c) it is able to generate a set of alternative good solutions in a reasonable time period; (d) it does not require any fine-tuning process; and (e) it can be easily executed in parallel following an agent-based simulation approach.

The SR-GCWS-CS algorithm[1]

Recent advances in computer processing power and in the development of high-quality pseudo-random number generators (RNGs) (L'Ecuyer, 2006) might have opened a new perspective regarding the use of probabilistic approaches in combinatorial problems. To test how simulation-based approaches can be used to improve existing heuristics and even push them to new efficiency levels, we combined MCS techniques with one of the best-known classical heuristics for the CVRP, namely the CWS heuristics. In particular, we selected the parallel version of this heuristic as it usually offers better results than the corresponding sequential one.

Combining Monte Carlo simulation with the CWS algorithm

One of the main ideas of the presented algorithm is to introduce some biased random behaviour within the CWS heuristic to perform a search process inside the space of feasible solutions. Each of these feasible solutions consists of a set of roundtrip routes from the depot that, altogether, satisfy all demands of the nodes by visiting and serving all of them exactly once. As stated in the previous section, at each step of the solution-construction process, the CWS algorithm always chooses the edge with the largest savings value (greedy behaviour). The presented approach, instead, assigns a selection probability to each edge in the savings list. Moreover, this probability should be coherent with the savings value associated with each edge, that is, edges with larger savings will be more likely to be selected from the list than those with smaller savings. Finally, this selection process should be done without introducing too many parameters in the algorithm—otherwise, it would be necessary to perform a fine-tuning process, which tends to be non-trivial and time-consuming. To reach all those goals, we use different geometric statistical distributions during the randomized CWS solution-construction process: every time a new edge is selected from the list of available edges, a value α is randomly selected from a uniform distribution in (a, b), where $0 < a \leqslant b < 1$. This parameter α defines the specific geometric distribution that will be used to assign

[1] SR-GCWS-CS stands for SimORouting's Generalized CWS with Cache and Splitting.

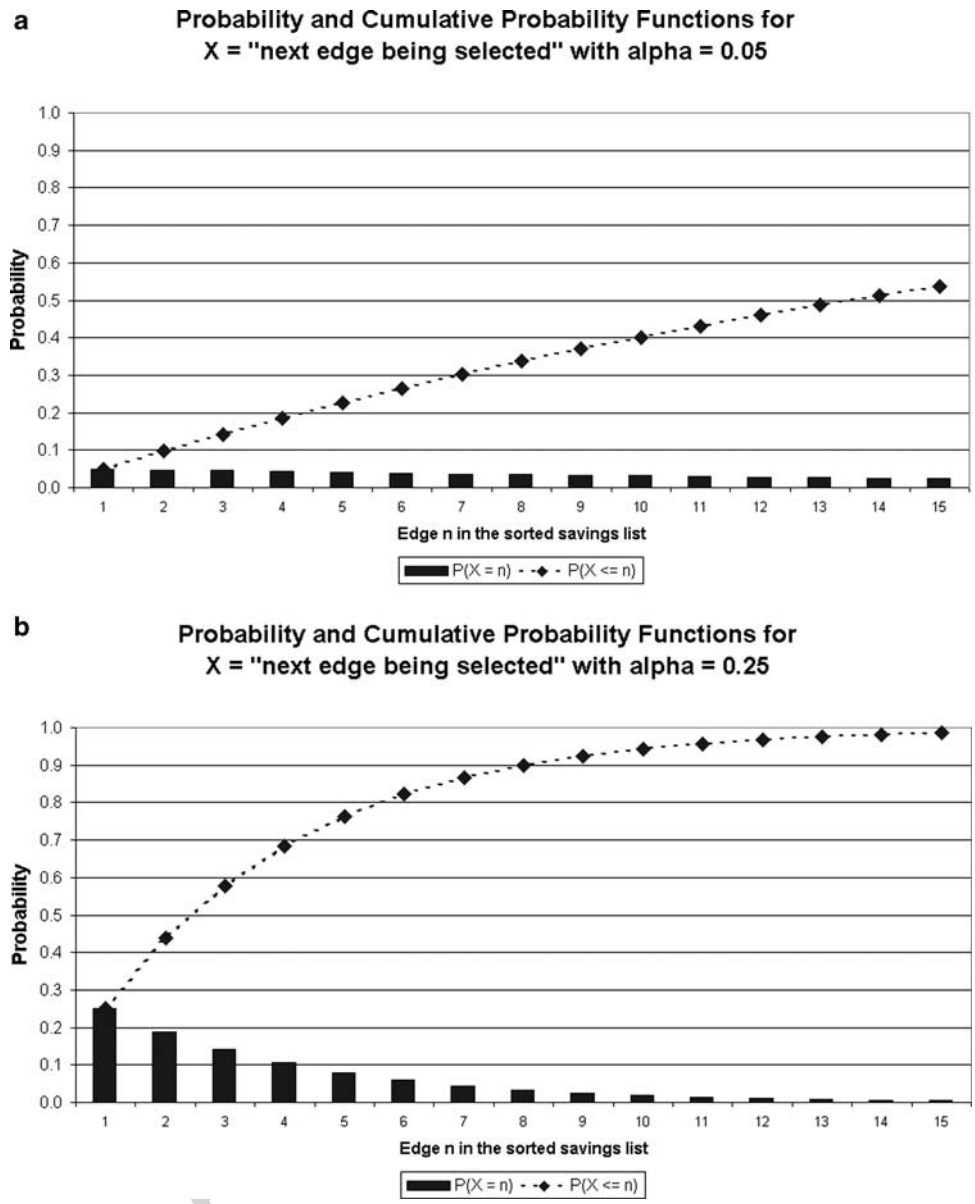

Figure 1 Geometric distributions for $\alpha = 0.05$ and $\alpha = 0.25$.

exponentially diminishing probabilities to each eligible edge according to its position inside the sorted savings list. That way, edges with higher savings values are always more likely to be selected from the list, but the exact probabilities assigned are variable and they depend upon the concrete distribution selected at each step (Figure 1). Using the values $a = 0.05$ and $b = 0.25$ pseudo-optimal solutions have been obtained for all tested instances. By iterating this solution-construction process in a personal computer, different randomized CWS solutions—some of them outperforming by far the original CWS solution—can be obtained in just a few seconds.

Introduction of a cache of pseudo-optimized routes

The iterative process described above can be enriched with a simple but efficient learning mechanism that altogether achieves a faster initial rate of convergence to the optimal solutions. The basic idea of this learning mechanism is described next: for each generated route, the best-known order to travel among the nodes that constitute that route is stored in cache memory—this could be done by using an array or, as in our case, by using a hash table which allows for a quick access to its elements. This cache is constantly updated whenever a better order with a lower cost is found for a given set of nodes—that is, whenever a more efficient route containing exactly the same set of nodes is discovered. At the same time, during the construction process the routes contained in this cache are re-used whenever possible to improve newly generated solutions. As the cache memory stores the best visiting order found for each set of nodes that can be served by a single vehicle, most new solutions will benefit from using

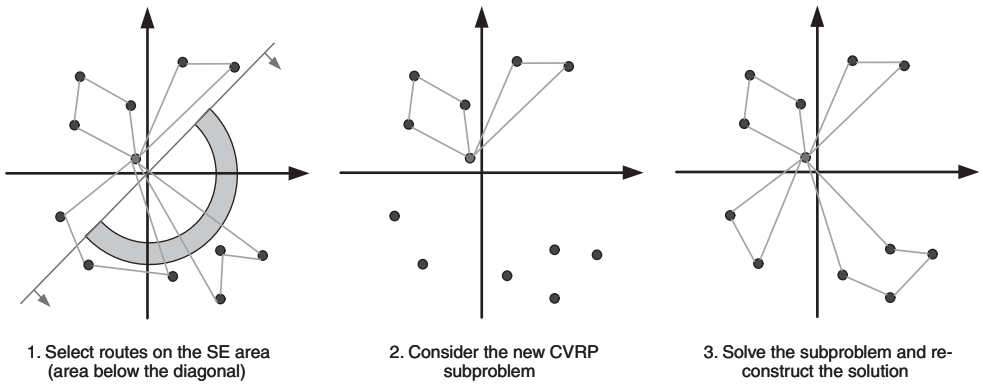

Figure 2 Using splitting policies to reduce the problem dimension.

this information. Notice that without this learning mechanism, some newly generated solutions could improve some parts (routes) of the best solution found so far but, at the same time, include some inefficient routes that deteriorate the global score—thus avoiding the achievement of a new best-known solution. Notice also that this cache mechanism allows the algorithm to 'learn from the past experience' in the sense that the average quality of newly generated solutions will increase with computational time. According to our experiments, the use of this cache methodology has implied a significant reduction of the computing times needed to obtain quasi-optimal solutions for most tested instances.

Splitting policies

Another important idea behind our approach is the use of splitting or divide-and-conquer policies to reduce the original problem size. The goal here is to use some proximity-based criterion to divide the original set of nodes into disjoint subsets and then to solve each of these subsets by applying the same methodology described before (ie, a combination of MCS with CWS enriched with a cache memory). Different splitting policies can be used, but the basic principle should always be the same: once a randomized CWS solution has been found, we use a proximity or geometric criterion to select a subset of routes and their corresponding nodes. Then, we reapply the iterative construction process to this subset in order to find a set of different routes with a better cost. The proximity criterion used in this work is based on the position of the geometric centre of each route with respect to the position of the global solution geometric centre—that is, with respect to all the nodes' geometric centre. The geometric centre of a given set of nodes is the average of the x and y coordinates of those nodes. Notice that the geometric centre is considered as the reference point instead of the depot. This makes it possible to obtain a balanced splitting even for those VRP instances where the depot is located in an extreme position with respect to the rest of the nodes. Even when many other approaches are possible, for the experiments described in this paper a total of 56 splitting policies were defined and used. All these policies are based on the aforementioned proximity criterion, which makes use of the basic Euclidean geometry to cover different regions of the nodes scatterplot. Figure 2 shows an example of one of the splitting policies being used. The solution space is divided into two regions, and a different VRP is solved on each subspace. The final solution merges the partially solved VRP sub-problems.

The 56 splitting policies used (avoiding symmetries) are summarized in Figure 3, and can be defined as follows:

- The region covering the first 3 quadrants (Figure 3(a)), and the 3 regions resulting from a rotation of $\pi/2$ (Figure 3(b)) in clockwise direction (regions 1–4).
- The same initial 4 regions rotated clockwise $\pi/4$ radians (Figure 3(c)) (regions 5–8).
- The same initial 4 regions rotated clockwise $\pi/8$ radians (Figure 3(d)) (regions 9–12).
- The same initial 4 regions rotated clockwise $3\pi/8$ radians (Figure 3(e)) (regions 13–16).
- The region covering the first 2 quadrants and an additional half quadrant (Figure 3(f)), with their corresponding $\pi/2$ rotations (regions 17–20).
- The region covering the first 2 quadrants and an additional half quadrant, with their corresponding $\pi/4$ possible rotations (regions 17–24).
- The 8 regions resulting from rotating the previous ones $\pi/8$ radians (regions 25–32).
- The regions covering 2 quadrants (Figure 3(g)) and the corresponding 4 rotations of $\pi/2$ radians, 4 rotations of $\pi/4$ and 8 rotations of $\pi/8$ radians (regions 33–48).
- Finally, 8 additional regions constructed from the opposite quadrants (Figure 3(h)) and their 2 rotations of $\pi/2$, 2 of $\pi/4$ and 4 of $\pi/8$ radians (regions 49–56).

Notice also that the centre of the regions' partition is located at the mean point of the nodes in the VRP problem, favouring a balanced splitting in the proposed algorithm. As a final remark, it is important to highlight that the described policies constitute just an example of possible divide-and-conquer

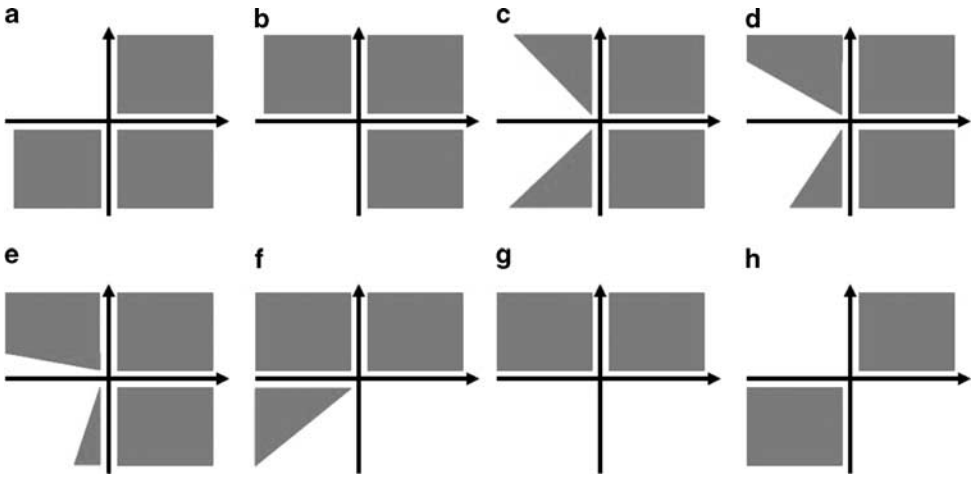

Figure 3 Base splitting models of the policies defined.

```
procedure SR-GCWS-CS(vrpNodes, vrpConstraints, algParameters, costsMatrix)
// algParameters include rng, nSols, nIter and nIterPerSplit
1   savingsList = makeSavingsList(vrpNodes, costsMatrix);
2   cwsSol = constructCWSSol(vrpNodes, costsMatrix, savingsList, vrpConstraints);
3   while stopping criteria not satisfied do // it depends on nIter
4     vrpSol = constructRandomSol(vrpNodes, costsMatrix, savingsList,
          vrpConstraints, rng);
5     vrpSol = improveSolUsingRoutesCache(vrpSol, costsMatrix);
6     if vrpSol outperforms cwsSol then // vrpSol is a promising sol
7         vrpSol = improveSolUsingSplitting(vrpNodes, costsMatrix,
             savingsList, vrpConstraints, rng, nIterPerSplit, vrpSol, rCache);
8         bestSols = updateBestSolsList(vrpSol, bestSols, nSols);
9     fi
10  elihw
11  return bestSols;
end
```

Figure 4 Main procedure of the SR-GCWS-CS algorithm.

strategies and, therefore, other proximity-based strategies and policies could also be explored in future research projects.

The algorithm pseudo-code

In this section, the main parts of the SR-GCWS-CS algorithm, which implements the main ideas highlighted in the previous section, are discussed. Figure 4 shows the main procedure, which drives the solving methodology.

The algorithm receives as input, the nodes to be served, the set of constraints, the costs matrix and the algorithm parameters, including the random number generator (`rng`), the number of best solutions to save (`nSols`) and the number of first- and second-level iterations to run (`nIter` and `nIterPerSplit`, respectively). Then, the savings matrix is calculated and a savings list is constructed and sorted. The resulting list contains the potential edges to be selected sorted by their associated savings. Next, an initial solution is obtained applying the CWS heuristics (parallel version). The costs associated with this solution will be used as an upper bound limit for the costs of what we will consider a good solution. It is at this point when we start the first-level iterative process to generate hundred or thousands of new solutions outperforming the CWS. At each first-level iteration, a new solution is constructed by using the randomized CWS heuristic (Figures 5 and 6); then this new randomized solution is processed by the cache procedure (Figure 7), which uses cache best results from previous iterations to improve (if possible) the current randomized solution. If the resulting solution outperforms the CWS heuristic, it is considered a promising solution and it is then processed by the splitting procedure (Figure 8); this splitting procedure tries to improve it by first considering different subsets of routes (ie by reducing the problem dimension) and then applying a second-level iterative process over each of these subsets. This second-level iterative process, in turn, makes use of both the randomized CWS and the cache procedures. At the end of each first-level iteration, the resulting solution is stored, if appropriate, in a sorted array of best solutions found so far.

Figure 5 shows some details of the randomized CWS process: this process starts with the initial solution proposed by the CWS heuristics, and then adds a randomized edge-selection before proceeding with the merging of existing routes if possible. Accordingly, Figure 6 shows some details

```
procedure constructRandomSol(nodes, cMatrix, sList, constraints, rng)
1   effList = copyList(sList);
2   sol = constructInitialSol(nodes, cMatrix); // sol = {(0,i,0) / i in nodes}
3   while effList contains edges do
4      e = selectEdgeAtRandom(effList, rng);
5      iNode = getOrigin(e);
6      jNode = getEnd(e);
7      iR = getRoute(iNode, sol);
8      jR = getRoute(jNode, sol);
9      if all CWS route-merging conditions are satisfied (see constraints) then
10        sol = mergeRoutesUsingEdge(e, iR, jR, sol); // see CWS heuristic
11     fi
12     deleteEdgeFromList(e, effList);
13  elihw
14  return sol;
end
```

Figure 5 Randomization of the CWS heuristics.

```
procedure selectEdgeAtRandom(list, rng)
1   beta = generateRandomNumber(rng, a, b); // e.g.: a = 0.06 and b = 0.22
2   randomValue = generateRandomNumber(rng, 0, 1);
3   n = 0;
4   cumulativeProbability = 0.0;
5   for each edge e in sorted list
6      eProbability = beta * (1 - beta)^n; // use geometric distribution
7      cumulativeProbability += eProbability;
8      if randomValue < cumulativeProbability then
9         return e;
10     else
11        n = n + 1;
12     fi
13  rof
14  k = generateIntegerRandomNumber(rng, 0, listSize); // k in [0, listSize)
15  return getEdgeAtPosition(k);
end
```

Figure 6 Random edge-selection using a geometric distribution.

```
procedure improveSolUsingRoutesCache(sol, cMatrix, rCache)
1   table = makeHashTable();
2   for each route r in sol
3      hashCode = getRouteHashCode(r);
4      if hashCode already in table then
5         routeInTable = getRouteInHashTable(rCache, hashCode);
6         if routeInTable outperforms r then
7            r = routeInTable;
8            sol = updateRouteInSol(r, sol);
9         else
10           r = improveNodesOrder(r, cMatrix); // aims to avoid knots in r
11           rCache = updateRouteInHashTable(rCache, hashCode, r);
12        fi
13     else
14        r = improveNodesOrder(r, cMatrix);
15        rCache = putRouteInHashTable(rCache, hashCode, r);
16     fi
17  rof
18  return sol;
end
```

Figure 7 A simple learning mechanism based on a cache of routes.

of this random selection process, which as explained before, is based on the use of geometric distributions.

Figure 7 shows some details of the cache procedure behaviour, which usually represents a quick solution-improvement process. Notice how the hash table always keeps the best-known order to visit a set of customers in a route. As already discussed before, this could be considered as a learning mechanism that makes the algorithm more efficient as more new solutions are generated. A hash table is used here for computational efficiency reasons.

```
procedure improveSolUsingSplitting(vrpNodes, costsMatrix,savingsList,
   vrpConstraints, rng, nIterPerSplit, vrpSol, rCache)
1   vrpCenter = calcGeometricCenter(vrpNodes); // (x-bar, y-bar) of nodes
2   sol = vrpSol; // initial sol
3   while a new splitting policy is available then
4      allRoutes = getRoutes(sol);
5      routesCenters = calcRoutesGeometricCenters(allRoutes);
6      frontRoutes = selectFrontRoutes(allRoutes, routesCenters, policy);
7      backRoutes = substractRoutesFromList(frontRoutes, allRoutes);
8      frontSubSol = makeSubSol(frontRoutes);
9      backSubSol = makeSubSol(backRoutes);
10     frontNodes = getNodes(frontSubSol);
11     splitSavingsList = makeSavingsList(frontNodes, savingsMatrix);
12     splitSavingsList = sortList(splitSavingsList);
13     while stopping criteria not satisfied (1<= iter <= nIterPerSplit) then
14        newSubSol = constructRandomSol(frontNodes, costsMatrix,
             splitSavingsList, constraints, rng);
15        newSubSol = improveSolUsingRoutesCache(newSol, costsMatrix, rCache);
16        if newSubSol outperforms frontSubSol then
17           frontSubSol = newSubSol;
18        fi
19     elihw
20     newSol = unifySubSols(frontSubSol, backSubSol);
21     if newSol outperforms sol then
22        sol = newSol;
23     fi
24  elihw
25  return sol;
end
```

Figure 8 Applying the splitting approach to reduce the problem dimension.

Finally, Figure 8 provides some insight of the splitting approach designed to reduce the problem dimension and, therefore, the computational effort necessary to obtain quasi-optimal solutions.

As explained in the previous section, for each complete or 'first-level' solution provided by the randomized CWS, splitting strategies can be used to select different subsets of routes from it. For each of these subsets, the associated nodes can then be used to define a new reduced or 'second-level' CVRP instance. Obviously, the same methodology used for the first-level instance can be now applied for each of these second-level instances: the corresponding savings list is obtained and sorted, and a second-level randomized iteration process is started. All in all, this two-level iterative strategy can be considered as a divide-and-conquer methodology that aims to improve the overall solution by solving reduced instances of the original problem.

Experimental results

The algorithm described in this paper has been implemented as a Java application and also as a multithread C program for faster execution. At the core of these implementations, some state-of-the-art pseudo-random number generators are used. In particular, in the case of the Java implementation we have used some classes from the SSJ library (L'Ecuyer, 2002), among them, the subclass GenF2W32, which implements a high quality generator (from a statistical point of view) with a period value equal to 2^{800}-1. Using a pseudo-random number generator with such an extremely long period is especially useful when performing an in-depth random search of the solutions space. In our opinion, the use of such a long-period RNG has other important advantages: the algorithm can be easily parallelized by splitting the RNG sequence in different streams and using each stream in different processes, which can be especially useful when dealing with large-scale instances. This can be an interesting field to explore in future works, given the current performance trend in distributed and parallel computing using multi-core processors and Graphical Process Units (GPUs).

To verify the feasibility of our approach and its efficiency as compared with other existing methodologies, a total of 33 classical CVRP benchmark instances were selected from three reference web sites: http://www.branchandcut.org (source 1), http://neo.lcc.uma.es/radi-aeb/WebVRP (source 2), and http://www.coin-or.org (source 3). These sites contain detailed information on a large number of benchmark instances. The selection process was based on the following criteria: (a) instances from different sets (A, B, E, F, M and P) were used, (b) only instances offering complete information (eg specific routes in best-known solution) were considered, (c) instances with less than 22 nodes were avoided, since they can be easily optimized by using exact methods. The selected benchmark files are shown in Table 1. These instances differ in the number of nodes and also in the location of the depot with respect to the customers, that is, in some cases the depot occupies a central position in the scatterplot of nodes while in others the depot is located at one corner. Some instances characterized by having clusters of nodes have also been considered.

Table 1 Results for 33 standard benchmarks (multi-thread C implementation)

Instance	*# nodes*	*vCap*	*CWS Sol.*	*Gap BSK-CWS (%)*	*Best-known Sol.*	*BKS (int)*	*# routes BKS*	*Source*	*Gap BKS-OBS (%)*	*Our Best Sol.*	*# routes OBS*	*Time (s)*
A-n32-k5	32	100	843.69	7.09	787.81	784	5	2	−0.09	787.08	5	1
A-n38-k5	38	100	768.14	4.63	734.18	730	5	2	−0.03	733.95	5	1
A-n45-k7	45	100	1 199.98	4.59	1 147.28	1 146	7	1	−0.04	1 146.77	7	1
A-n55-k9	55	100	1 099.84	2.36	1 074.46	1 073	9	2	0.00	1 074.46	9	1
A-n60-k9	60	100	1 421.88	4.87	1 355.80	1 354	9	1	0.00	1 355.80	9	1
A-n61-k9	61	100	1 102.23	6.08	1 039.08	1 034	9	2	0.00	1 039.08	9	2
A-n65-k9	65	100	1 239.42	4.89	1 181.69	1 174	9	2	0.00	1 181.69	9	2
A-n80-k10	80	100	1 860.94	5.35	1 766.50	1 763	10	1	0.00	1 766.50	10	178
B-n50-k7	50	100	748.80	0.54	744.78	741	7	1	−0.07	744.23	7	1
B-n52-k7	52	100	764.90	1.98	750.08	747	7	1	−0.02	749.96	7	1
B-n57-k9	57	100	1 653.42	3.10	1 603.63	1 598	9	1	−0.08	1 602.28	9	1
B-n78-k10	78	100	1 264.56	2.87	1 229.27	1 221	10	1	−0.11	1 227.90	10	9
E-n22-k4	22	6 000	388.77	3.59	375.28	375	4	2	0.00	375.28	4	1
E-n30-k3	30	4 500	534.45	−0.25	535.80	534	3	2	−5.75	505.01	4	1
E-n33-k4	33	8 000	843.10	0.52	838.72	835	4	2	−0.13	837.67	4	1
E-n51-k5	51	160	584.64	11.37	524.94	521	5	1	−0.06	524.61	5	1
E-n76-k7	76	220	737.74	7.29	687.60	682	7	2	0.00	687.60	7	7
E-n76-k10	76	140	900.26	7.51	837.36	830	10	1	−0.25	835.28	10	231
E-n76-k14	76	100	1 073.43	4.55	1 026.71	1 021	14	1	−0.22	1 024.40	15	32
F-n45-k4	45	2 010	739.02	1.99	724.57	721	4	3	−0.14	723.54	4	5
F-n72-k4	72	30 000	256.19	2.97	248.81	237	4	3	−2.75	241.97	4	2
F-n135-k7	135	2 210	1 219.32	4.16	1 170.65	1 159	7	1	−0.51	1 164.73	7	131
M-n101-k10	101	200	833.51	1.67	819.81	820	10	3	−0.03	819.56	10	0
M-n121-k7	121	200	1 068.14	2.20	1 045.16	1 034	7	1	−0.12	1 043.88	7	107
P-n22-k8	22	3 000	590.62	−1.80	601.42	603	8	3	−2.10	588.79	9	1
P-n40-k5	40	140	518.37	12.27	461.73	458	5	3	0.00	461.73	5	1
P-n50-k10	50	100	734.32	4.97	699.56	696	10	3	0.00	699.56	10	2
P-n55-k15	55	70	978.07	−1.35	991.48	989	15	3	−4.73	944.56	16	0
P-n65-k10	65	130	851.67	6.90	796.67	792	10	3	−0.13	795.66	10	2
P-n70-k10	70	135	896.86	8.05	830.02	827	10	1	−0.01	829.93	10	1
P-n76-k4	76	350	689.13	15.20	598.22	593	4	3	−0.01	598.19	4	88
P-n76-k5	76	280	698.51	9.99	635.04	627	5	3	−0.27	633.32	5	3
P-n101-k4	101	400	765.38	10.56	692.28	681	4	1	−0.14	691.29	4	153
Averages	63			4.87					−0.54			29

A standard personal computer, Intel® Core™2 Quad Q8200 CPU at 2.4 GHz and 2 GB RAM, was used to perform these tests. Results of these tests are summarized in Table 1, which contains the following information for each instance: number of nodes, vehicle capacity, costs given by the CWS solution, gap between the CWS solution and the best-known solution, current values for the best-known solution—both using real and integer (rounded) numbers—, associated number of routes, source web site, gap between the best-known solution and the best solution obtained using our algorithm, current value for our best solution, associated number of routes, and computation time used. Each instance was run 25 times and only best solutions are reported.

Discussion of results

First of all, notice that negative gaps in column 'Gap BKS-OBS' of Table 1 correspond to improvements with respect the best known so far solution. Care must be taken when analyzing the best known so far solutions, as most sources give these solutions as integer numbers, which have been obtained by rounding initial costs. From the detailed solution provided by those sources, we were able to calculate the corresponding real costs, which we compared against the costs obtained with our algorithm. Thus, from Table 1 it can be deduced that our algorithm is able to provide, in a reasonable time—just a few seconds for most instances—a virtually equivalent (or better) solution to the best solution known so far. Notice that in all 33 instances, our algorithm has been able to match or slightly improve (negative gaps) the value associated with the best known solution when this is calculated using real numbers. Moreover, this has been done in just a few seconds using the multithread C version of the algorithm. Notice also that in four instances (E-n30-k3, E-n76-k14, P-n22-k8, and P-n55-k15) the best solution provided by our algorithm uses one more route than the best-known

solution. Even when direct comparisons between two different algorithms are always a delicate issue—especially if they have been implemented and executed using different computer environments—it is possible to make a preliminary comparison between our results and those obtained using other probabilistic approaches. In particular, our approach outperforms the Ant Colony Optimization (ACO) procedure developed by Mazzeo and Loisseau (2004) in all four instances commonly reported (E-n51-k5, E-n76-k10, M-n101-k10, and M-n121-k7, which results are given as rounded integers in the case of the ACO approach). In summary, for the set of instances considered, which average number of nodes is 63, our algorithm provided a negative mean gap of 0.54% in an average time of 29 seconds. In our opinion, this can be considered a remarkable result and one that encourages us to develop future work regarding the possibilities of this algorithm in solving large-scale VRP instances.

As described before, our approach makes use of an iterative process to generate a set of random feasible solutions. According to the experimental tests, in the case of small- and medium-size instances, each iteration is completed in just a few milliseconds by using a standard computer. By construction, odds are than the generated solution outperforms the CWS heuristic. This means that the presented approach provides, almost immediately (even for the larger instances considered in our study), a feasible solution which outperforms the CWS heuristic in aprioristic costs. Moreover, as verified by tests, hundreds or even thousands of alternative solutions outperforming the CWS heuristic can be obtained after some seconds of computation. Each of them has different attributes in non-aprioristic costs, workload balance, visual attractiveness, etc. Thus a list of alternative solutions can be constructed, allowing the decision-maker to filter through the list according to different criteria. This list offers the decision-maker the possibility to choose, among different solutions with similar aprioristic costs, the one which best fulfils his or her preferences according to his or her utility function. Figure 9 shows a typical example of a non-attractive solution characterized by intersecting routes. In some cases, the decision-maker might want to avoid these solutions if other alternatives are available. Also, Figure 9 shows a typical situation to avoid due to the existence of highly unbalanced routes. Even when this could be an optimal solution according to the objective function defined a priori, it seems unlikely that the decision-maker uses it if other alternatives, with similar costs, are available.

Another important point to consider here is the relative simplicity of the presented algorithm. This is quite interesting in our opinion, since according to Kant *et al* (2008) some of the most efficient heuristics and metaheuristics are not used in practice because of the difficulties they present when dealing with real-life problems and restrictions. On the contrary, simple hybrid approaches like ours tend to be more flexible and, therefore, they seem more appropriate to deal with real restrictions and dynamic work conditions. Notice also that the presented approach seems to work well in all tested instances without requiring any special fine-tuning or set-up process.

All in all, according to the experimental results, it seems acceptable to say that the hybrid algorithm presented here can be considered as an alternative to other metaheuristics when looking for a pseudo-optimal solution for a CVRP instance in a reasonable time, especially if difficult parameter fine-tuning processes must be avoided—as it happens in most real-life scenarios.

Finally, it is convenient to highlight that the introduced algorithm can be used beyond the CVRP scenario: with little effort, similar hybrid algorithms based on the combination of Monte Carlo simulation with already existing heuristics can be developed for other routing problems and, in general, for other combinatorial optimization problems. In our opinion, this opens a new range of potential applications that could be explored in future works.

Comparison between our approach and GRASP

Clearly, some important characteristics of the algorithm presented in this paper can be identified in the previous description of those algorithms belonging to the GRASP family: (a) it is a multi-start process with a construction phase and a (kind of) local-search phase; (b) during the construction phase one element at a time is selected from a list of candidates using a probabilistic approach, etc. Therefore, to some extend it is reasonable to consider our algorithm as a member of the large GRASP family, especially if we refer to the fuzzy concept of Hybrid GRASP algorithms (Resende, 2008). However, in our opinion the algorithm presented in this paper has some characteristics that are quite singular and make it differ from classical GRASP algorithms, for example: (a) it could be argued that the SR-GCWS-CS does not use a local-search phase but instead two nested constructive phases (previously referred to as first-level and second-level iterative processes); (b) the probabilistic component of our algorithm is provided by sampling from a well-known statistical distribution, which defines a strong relationship of our approach with simulation techniques; (c) the SR-GCWS-CS is not adaptive in the sense that the savings associated with each component of the list remain the same throughout the construction process; (d) our approach makes use of some peculiar splitting and memory-based techniques which are not frequent in typical GRASP (although Hybrid GRASP algorithms can consider also memory and learning techniques).

Even when GRASP algorithms have been successfully applied to a wide range of combinatorial problems (Festa and Resende, 2009b), to the best of our knowledge, these have traditionally underperformed other metaheuristics (eg Genetic Algorithms, Tabu Search, etc) in solving intensively-tested CVRP benchmarks. Although some efficient GRASP algorithms have been reported for solving other less explored VRP variants (eg VRPTW), in the CVRP arena only the

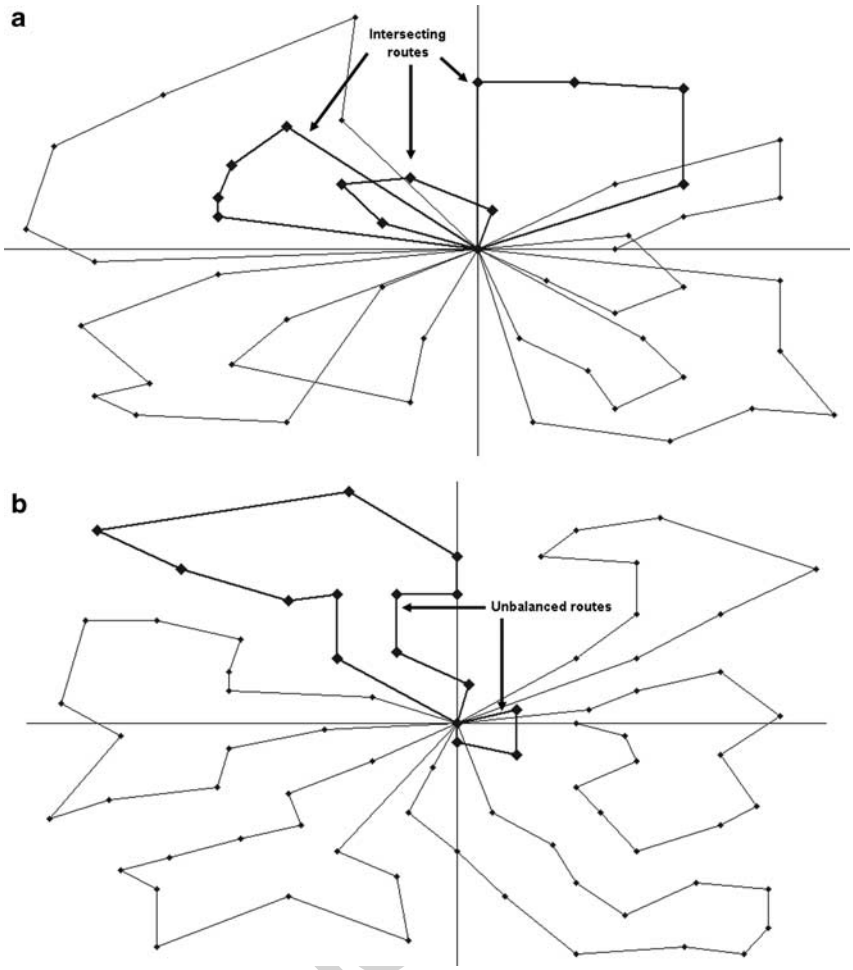

Figure 9 Typical examples of non-realistic pseudo-optimal solutions.

GRASP algorithm from Baker and Carreto (2003) seems to provide efficient results when compared with state-of-the-art metaheuristics (Gendreau *et al*, 2008). Unfortunately, their visual interactive system CRUISE was only tested against a heterogeneous set of 14 benchmark problems (only some of them being purely CVRP), and the reported results showed a positive average gap of 0.84% with respect the best-known solutions after a relatively large execution time (about 15 minutes per instance using a Pentium II computer running at 300 MHz). Moreover, the CRUISE algorithm requires from a non-trivial fine-tuning process—for example, the user has to decide about the maximum length of the restricted candidate list and also to define a weight for each customer on the list so that 'priority is given to customers with larger demands'. In our opinion, this is a kind of trial-and-error process that can be time-consuming and could require from some human expertize. Therefore, should our algorithm be considered a member of the vast Hybrid GRASP family, it would probably be the first GRASP-related algorithm able to compete with other state-of-the-art metaheuristics in the CVRP arena.

Future work

We are currently working on three different research lines. First, we are developing a parallel implementation of the SR-GCWS-CS algorithm to significantly accelerate its execution speed when dealing with large-scale instances. Having this purpose, we are implementing the algorithm using the NVIDIA's CUDA architecture, so that it benefits from the outstanding processing capabilities of modern Graphical Processing Units. Notice that because of its design, the algorithm could be easily parallelized by simply executing it in different threads, with each thread running under a different seed value for the associated random number generator that is, using an agent-based simulation approach. In this context, there are at least three interesting alternatives to explore: (a) considering each thread execution as an independent agent owning a private hash table (cache memory); (b) sharing a centralized hash table among all threads; and (c) using several distributed hash tables which communicate among themselves to share information at certain time periods. Options

(a) and (b) are directly related to the use of GPUs or parallel computing in shared-memory platforms, whereas option (c) might be more related to the use of distributed-memory platforms like Grid, Cloud or Volunteer computing environments. Our second research line is focused on incorporating complete search—for example, constraint programming, mixed integer linear programming, etc—to the optimization of the single vehicle routes stored in the hash table. This may help to improve future global solutions by piecing together these optimized routes stored in the hash table in order to construct a puzzle-like solution. Finally, the third line of our research is concerned with exploring potential applications of the presented algorithm in solving other routing and scheduling problems, as most of them are likely to have classical heuristics that could be efficiently combined with Monte Carlo simulation to obtain quasi-optimal solutions.

Conclusions

In this paper we have presented the SR-GCWS-CS algorithm, a relatively simple and yet flexible procedure for solving the Capacitated Vehicle Routing Problem. This algorithm, which does not require any particular fine-tuning or configuration process, combines the classical Clarke and Wright heuristic with Monte Carlo simulation, a memory-based learning mechanism and splitting (divide-and-conquer) strategies. Results show that the SR-GCWS-CS can be a real alternative to other metaheuristics since it is able to provide top-quality solutions to most tested benchmarks in reasonable times. Moreover, as the algorithm follows a constructive approach, it can generate hundreds or even thousands of alternative good solutions, thus offering the decision-maker the possibility to apply different non-aprioristic criteria when selecting the solution that best fits his or her utility function. Finally, it is important to highlight that the SR-GCWS-CS algorithm may be a fine framework for the development of similar algorithms for other complex combinatorial problems in the routing arena as well as in some other research fields.

Acknowledgements—This work has been partially supported by the IN3-UOC Knowledge Community Program under grant HAROSA KC (http://dpcs.uoc.edu) and by the project TRA2006-10639 from the Spanish Ministry of Science and Technology. The authors would also like to thank NVIDIA Inc. by their kindly support to our research.

References

Baker B and Carreto C (2003). A visual interactive approach to vehicle routing. *Comput Opns Res* **30**: 321–337.

Bastiaans R (2000). Reducing traffic jams, pollution and accidents. *RTD Info*, 25, http://ec.europa.eu/research/rtdinfo/en/25/05.html, accessed 17 November 2009.

Battarra M, Golden B and Vigo D (2008). Tuning a parametric Clarke-Wright heuristic via a genetic algorithm. *J Opl Res Soc* **59**: 1568–1572.

Beasley J (1981). Adapting the savings algorithm for varying inter-customer travel times. *Omega* **9**: 658–659.

Berger J and Barkaoui M (2003). A hybrid genetic algorithm for the capacitated vehicle routing problem. In: Cantó-Paz E (ed). *Proceedings of the International Genetic and Evolutionary Computation Conference—GECCO03*. LNCS 2723, Springer-Verlag, Illinois, Chicago, USA, pp 646–656.

Boudia M, Louly MAO and Prins C (2007). A reactive GRASP and path relinking for a combined production-distribution problem. *Comput Opns Res* **34**: 3402–3419.

Buxey GM (1979). The vehicle scheduling problem and Monte Carlo simulation. *J Opl Res Soc* **30**: 563–573.

Clarke G and Wright J (1964). Scheduling of vehicles from a central depot to a number of delivering points. *Opns Res* **12**: 568–581.

Cordeau JF, Gendreau M, Hertz A, Laporte G and Sormany JS (2004). New Heuristics for the Vehicle Routing Problem. In: Langevin A and Riopel D (eds). *Logistics Systems: Design and Optimization*. Kluwer Academic Publishers: Dordrecht (Hingham, MA).

Dror M and Trudeau P (1986). Stochastic vehicle routing with modified savings algorithm. *Eur J Opl Res* **23**: 228–235.

Duhamel, C Lacomme P, Prins C and Prodhon C (2010). A GRASP × ELS approach for the capacitated location-routing problem. *Comput Opns Res* **37**: 1912–1923.

Faulin J and Juan A (2008). The ALGACEA-1 method for the capacitated vehicle routing problem. *Int Trans Opl Res* **15**: 1–23.

Faulin J, Gilibert M, Juan A, Ruiz R, Vilajosana X (2008). SR-1: A simulation-based algorithm for the capacitated vehicle routing problem. In: Mason SJ, Hill R, Moench L and Rose O (eds). *Proceedings of the 2008 Winter Simulation Conference*, Miami, Florida, USA, December 7–10, pp. 2708–2716.

Feo TA and Resende MGC (1995). Greedy randomized adaptive search procedures. *J Global Optim* **6**: 109–133.

Fernández de Córdoba P, García Raffi LM, Mayado A and Sanchis JM (2000). A real delivery problem dealt with Monte Carlo techniques. *TOP* **8**: 57–71.

Festa P and Resende MGC (2009a). An annotated bibliography of GRASP—Part I: algorithms. *Int Trans Opl Res* **16**: 1–24.

Festa P and Resende MGC (2009b). An annotated bibliography of GRASP—Part II: applications. *Int Trans Opl Res* **16**: 131–172.

Gaskell TJ (1967). Bases for vehicle fleet scheduling. *Opl Res Quart* **18**: 281–295.

Gendreau M, Hertz A and Laporte G (1994). A tabu search heuristic for the vehicle routing problem. *Mngt Sci* **40**: 1276–1290.

Gendreau M, Potvin J-Y, Bräysy O, Hasle G and Lokketangen A (2008). Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In: Golden B, Raghavan S and Wasil E (eds). *The Vehicle Routing Problem: Latest Advances and New Challenges*. Operations Research/Computer Science Interfaces Series, Vol. 43. Springer: New York, pp 143–169.

Golden B, Raghavan S and Wasil EE (eds) (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer: New York, USA.

Gouvêa EF, Golbarg MC and Farias JPF (2007). GRASP with path-relinking for the TSP. In: Doerner KF, Gendreau M, Greistorfer P, Gutjahr WJ, Hartl R and Reimann M (eds). *Metaheuristics. Progress in Complex Systems Optimization*. Operations Research/Computer Science Interfaces Series, Vol. 39. Springer: New York, USA, pp 137–152.

Holmes RA and Parker RG (1976). A vehicle scheduling procedure based upon savings and a solution perturbation scheme. *Opl Res Quart* **27**: 83–92.

Juan A, Faulin J, Jorba J, Grasman S, and Barrios B (2008). SR-2: A hybrid intelligent algorithm for the vehicle routing problem. In: Xhafa F, Herrera F, Abraham A, Köppen M and Benitez JM (eds). *Proceedings of the 8th International Conference on Hybrid Intelligent Systems*, IEEE Computer Society: Barcelona, Spain, pp 78–83.

Juan A, Faulin J, Ruiz R, Barrios B, Gilibert M and Vilajosana X (2009). Using oriented random search to provide a set of alternative solutions to the capacitated vehicle routing problem. In: Chinneck J, Kristjansson B and Saltzman M (eds). *Operations Research and Cyber-Infrastructure*. Operations Research/Computer Science Interfaces Series, Vol. 47. Springer: New York: USA, pp 331–346.

Kant G, Jacks M and Aantjes C (2008). Coca-cola Enterprises optimizes vehicle routes for efficient product delivery. *Interfaces* **38**: 40–50.

Kumares C and Labi S (2007). *Transportation Decision Making: Principles of Project Evaluation and Programming*. Wiley: Hoboken, NJ.

Laporte G (2007). What you should know about the vehicle routing problem. *Nav Res Log* **54**: 811–819.

Law A (2007). *Simulation Modeling & Analysis*. McGraw-Hill: Boston.

L'Ecuyer P (2002). SSJ: A framework for stochastic simulation in Java. In: Yücesan E, Chen CH, Snowdon JL and Charnes JM (eds). *Proceedings of the 2002 Winter Simulation Conference*, pp 234–242.

L'Ecuyer P (2006). Random Number Generation. In: Henderson SG and Nelson BL (eds). *Handbooks in Operations Research and Management Science: Simulation*. Elsevier Science: Amsterdam, pp 55–81.

Mazzeo S and Loisseau I (2004). An ant colony algorithm for the capacitated vehicle routing. *Electron Notes Discrete Math* **18**: 181–186.

Mester D and Bräysy O (2007). Active-guided evolution strategies for the large-scale capacitated vehicle routing problems. *Comput Opn Res* **34**: 2964–2975.

Mole RH and Jameson SR (1976). A sequential route-building algorithm employing a generalised savings criterion. *Opl Res Quart* **27**: 503–511.

Nagata Y (2007). Edge assembly crossover for the capacitated vehicle routing problem. In: Cotta C and van Hemert J (eds). *Lecture Notes in Computer Science*. Springer-Verlag: Berlin Heidelberg, pp 142–153.

Paessens H (1988). The savings algorithm for the vehicle routing problem. *Eur J Opl Res* **34**: 336–344.

Poot A, Kant G and Wagelmans A (2002). A savings based method for real-life vehicle routing problems. *J Opl Res Soc* **53**: 57–68.

Prins C (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Comput Opns Res* **31**: 1985–2002.

Resende MGC (2008). Metaheuristic hybridization with greedy randomized adaptive search procedures. In: Chen Z and Raghavan S (eds). *TutORials in Operations Research. State-of-the-Art Decision-Making Tools in the Information-Intensive Age*. INFORMS: Hanover, MD, pp 295–319.

Taillard E (1993). Parallel iterative search methods for vehicle routing problems. *Networks* **23**: 661–673.

Tarantilis CD and Kiranoudis CT (2002). BoneRoute: An adaptative memory-based method for effective fleet management. *Ann Opns Res* **115**: 227–241.

Toth P and Vigo D (2002). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, pp 109–129.

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

Received March 2009;
accepted February 2010 after one major revision