Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Computers & Operations Research 31 (2004) 1985–2002

computers & operations research

www.elsevier.com/locate/dsw

A simple and effective evolutionary algorithm for the vehicle routing problem

Christian Prins*

LOSI, University of Technology of Troyes, BP 2060, 10010 Troyes Cedex, France

Abstract

The vehicle routing problem (VRP) plays a central role in the optimization of distribution networks. Since some classical instances with 75 nodes resist the best exact solution methods, most researchers concentrate on metaheuristics for solving real-life problems. Contrary to the VRP with time windows, no genetic algorithm (GA) can compete with the powerful tabu search (TS) methods designed for the VRP. This paper bridges the gap by presenting a relatively simple but effective hybrid GA. In terms of average solution cost, this algorithm outperforms most published TS heuristics on the 14 classical Christofides instances and becomes the best solution method for the 20 large-scale instances generated by Golden et al.

Scope and purpose

The framework of this research is the development of effective metaheuristics for hard combinatorial optimization problems met in vehicle routing. It is surprising to notice in the literature the absence of effective genetic algorithms (GA) for the vehicle routing problem (VRP, the main capacitated node routing problem), contrary to node routing problems with time windows or arc routing problems. Earlier attempts were based on chromosomes with trip delimiters and needed a repair procedure to get feasible children after each crossover. Such procedures are known to weaken the genetic transmission of information from parents to children. This paper proposes a GA without trip delimiters, hybridized with a local search procedure. At any time, a chromosome can be converted into an optimal VRP solution (subject to chromosome sequence), thanks to a special splitting procedure. This design choice avoids repair procedures and enables the use of classical crossovers like OX. The resulting algorithm is flexible, relatively simple, and very effective when applied to two sets of standard benchmark instances ranging from 50 to 483 customers.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Vehicle routing problem; Genetic algorithm

* Tel.: +33-325-71-56-41; fax: +33-325-71-56-49.
E-mail address: prins@utt.fr (C. Prins).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0305-0548(03)00158-8

1. Introduction

The vehicle routing problem (VRP) is defined on a complete undirected network $G=(V,E)$ with a node set $V=\{0,1,\ldots,n\}$ and an edge set E. Node 0 is a depot with m identical vehicles of capacity W, m can be fixed a priori or left as a decision variable. Each other node $i>0$ represents a customer with a non-negative demand q_i and each edge (i,j) has a non-negative travel cost $c_{ij}=c_{ji}$. The VRP consists of determining a set of m vehicle trips of minimum total cost, such that each trip starts and ends at the depot, each client is visited exactly once, and the total demand handled by any vehicle does not exceed W. In practice, customers are disseminated over a large road network and the cost on each edge is generally a mileage or a travel time. This real problem is contracted into a VRP by keeping the depot and client nodes and by computing the shortest path costs c_{ij} using Dijkstra's algorithm for instance (see [1], for an efficient implementation).

The VRP is NP-hard and some Euclidean instances with 75 nodes are not yet solved to optimality [2]. Therefore, heuristic algorithms are required for tackling real-life instances. The simplest and fastest methods are simple constructive heuristics like the classics proposed by Clarke and Wright, Mole and Jameson, and Gillett and Miller: see Laporte et al. [3] for a recent survey extended to tabu search algorithms, and Christofides et al. [4], for an earlier review. Metaheuristics provide much better solutions, especially on large-scale instances. Two excellent surveys for this very active research area are provided by Gendreau et al. [5] and by Golden et al. [6]. They show that the best metaheuristics for the VRP are powerful tabu search (TS) algorithms that easily outperform other metaheuristics like simulated annealing (SA), genetic algorithms (GA) and ant algorithms.

Concerning genetic algorithms, Van Breedam and Schmitt are worth citing. Van Breedam [7] measures the impact of various parameters on a GA and an SA and compares the two algorithms with a previously developed TS. His chromosomes contain one substring per trip and multiple copies of the depot used as trip delimiters. The crossover extends the classical order-based crossover PMX [8] and may produce infeasible children that are rejected. Van Breedam reports comparable results for the three algorithms, but the study is conducted on his own set of instances and no comparison is provided with metaheuristics already evaluated in the literature.

Schmitt [9,10] designed a GA with TSP-like permutation chromosomes (that is, without trip delimiters). The fitness is computed by scanning the chromosomes from their first customer onwards: a new trip is created each time the next customer cannot fit the current trip. Using the OX crossover [11], Schmitt evaluates his GA on the 14 classical instances proposed by Christofides et al. [4] with disappointing results: the GA easily outperforms the Clarke and Wright heuristic but cannot compete with some constructive heuristics combined with steepest descent improvement procedures.

In fact, several authors like Gendreau [5] underline the lack of competitive GA for the VRP. This situation seems abnormal since efficient GAs are available both for a simpler problem like the Traveling Salesman Problem or TSP [12] and for an extension like the VRP with time windows or VRPTW [13,14].

The goal of this paper is to remedy this anomaly by presenting a relatively simple but effective hybrid GA for a version of the VRP specified in Section 2. Section 3 underlines some key-points essential for the GA efficiency. The chromosomes and their evaluation are described in Section 4, while Sections 5, 6, 7 and 8 are, respectively, devoted to the crossover, to the local search used as mutation operator, to the population structure and to the GA iterations. A computational evaluation is presented in Section 9.

2. Problem treated and assumptions

The classical benchmarks used in Section 9 for testing comprise a few *distance-constrained VRP* (DVRP) instances: each customer is associated with a non-negative delivery cost d_i and the total cost of each trip (travel costs plus delivery costs) may not exceed a fixed limit L. This is why our GA is designed for the DVRP. Of course, it can solve the pure VRP by setting all d_i to 0 and L to a huge value.

$$\forall i,j,k \in V\colon\ c_{ik} + c_{kj} \geqslant c_{ij}, \tag{1}$$

$$\forall i \in V \setminus \{0\}\colon\ q_i \leqslant W, \tag{2}$$

$$\forall i \in V \setminus \{0\}\colon\ c_{0i} + d_i + c_{i0} \leqslant L. \tag{3}$$

In the sequel, the number of vehicles m is a decision variable, the costs are real numbers and the cost matrix satisfies the triangle inequality (1). To guarantee the existence of feasible solutions, the input data satisfy condition (2) (no demand must exceed vehicle capacity) and condition (3) (a return trip to each customer must be possible within the limit L). The goal is to minimize total cost of the trips. As we shall see, the GA is flexible enough to handle more complicated objectives.

3. Outline of key-points in the GA design

Our GA for the DVRP is based upon the following key-ideas, developed in the following sections:

K1. TSP-like permutation chromosomes, without trip delimiters.
K2. Exact fitness computation using a splitting procedure.
K3. Small population in which all individuals are distinct.
K4. Inclusion of good heuristic solutions in the initial population.
K5. Improvement procedure as mutation operator (hybrid GA).
K6. Incremental management of the population.
K7. Main exploration phase followed by a few short restarts.

Points 3–7 were already combined by Prins into a very efficient GA for the open-shop scheduling problem [15]. This GA outperforms two other GAs and three TS algorithms. It is still the only algorithm able to solve to optimality 30 instances with 100, 225 and 400 tasks proposed by Taillard, except one for which the best-known solution is improved. More recently, Lacomme et al. [16] have applied the same ideas, plus ideas 1–2 specific to vehicle routing, to the capacitated arc routing problem (CARP). There again, the resulting GA performs very well on classical benchmark problems: two thirds are solved to optimality and eight best-known solutions are improved. The lack of competitive GA was the main motivation for trying these successful ideas on the DVRP.

4. Chromosomes and evaluation

4.1. Principles

Like in most GAs for the TSP, a chromosome simply is a sequence (permutation) S of n client nodes, *without trip delimiters*. It may be interpreted as the order in which a vehicle must visit all customers, if the same vehicle performs all trips one by one. Contrary to Schmitt, who cuts S into trips sequentially, we use an *optimal splitting procedure Split*, detailed in 4.2, to get the best DVRP solution respecting the sequence. The fitness $F(S)$ of S is the total cost of this solution.

This kind of method was first put forward by Beasley [17] as the second phase of a route-first, cluster-second heuristic for the VRP. The first phase computes a giant TSP tour on all customers, by relaxing vehicle capacity and maximum tour length. Apart from a theoretical interest for proving some worst-case performance ratios, the method has never superseded more traditional VRP heuristics.

The situation is different for a GA. First, *there exists at least one optimal chromosome* (consider any optimal DVRP solution and concatenate the lists of nodes of its trips). Second, *if a crossover generates such a chromosome, the corresponding optimal DVRP solution can always be retrieved with Split*. Third, *the task of finding the best chromosome is left to the intrinsic parallelism of the GA*.

4.2. Algorithm

Split works on an auxiliary graph $H = (X, A, Z)$. X contains $n+1$ nodes indexed from 0 to n. A contains one arc (i, j), $i < j$, if a trip visiting customers S_{i+1} to S_j is feasible in terms of load (condition (4)) and cost (condition 5)). The weight z_{ij} of (i, j) is equal to the trip cost.

$$\forall (i,j) \in A\colon \sum_{k=i+1}^{j} q_{S_k} \leqslant W, \tag{4}$$

$$\forall (i,j) \in A\colon z_{ij} = c_{0,S_{i+1}} + \sum_{k=i+1}^{j} (d_{S_k} + c_{S_k,S_{k+1}}) + d_{S_j} + c_{S_j,0} \leqslant L. \tag{5}$$

An optimal DVRP solution for S corresponds to a min-cost path μ from 0 to n in H. This evaluation is reasonably fast because H is circuitless, $|A| = O(n^2)$, and the node numbering provides a natural topological ordering: in that case, μ can be computed in $O(n^2)$ using Bellman's algorithm [1]. The algorithm is faster when the minimal demand $q_{\min}$ is large enough: since a trip cannot visit more than $b = \lfloor W/q_{\min} \rfloor$ customers, the complexity becomes $O(nb)$.

The top of Fig. 1 shows a sequence $S = (a, b, c, d, e)$ with $W = 10$, $L = \infty$ and demands in brackets. H in the middle contains for instance one arc ab with weight 55 for the trip $(0, a, b, 0)$. μ has three arcs and its cost is 255 (bold lines). The lower part gives the resulting VRP solution with three trips.

The algorithm of Fig. 2 is a version in $O(n)$ space that does not generate H explicitly. It computes two labels for each node $j = 1, 2, \ldots, n$ of X: V_j, the cost of the shortest path from node 0 to node j in H, and P_j, the predecessor of j on this path. The *repeat* loop enumerates all feasible sub-sequences $S_i \ldots S_j$ and directly updates V_j and P_j. The required fitness $F(S)$ is given at the end by V_n. For a given i, note that the incrementation of j stops when L is exceeded: no feasible trip is discarded since the triangle inequality holds.

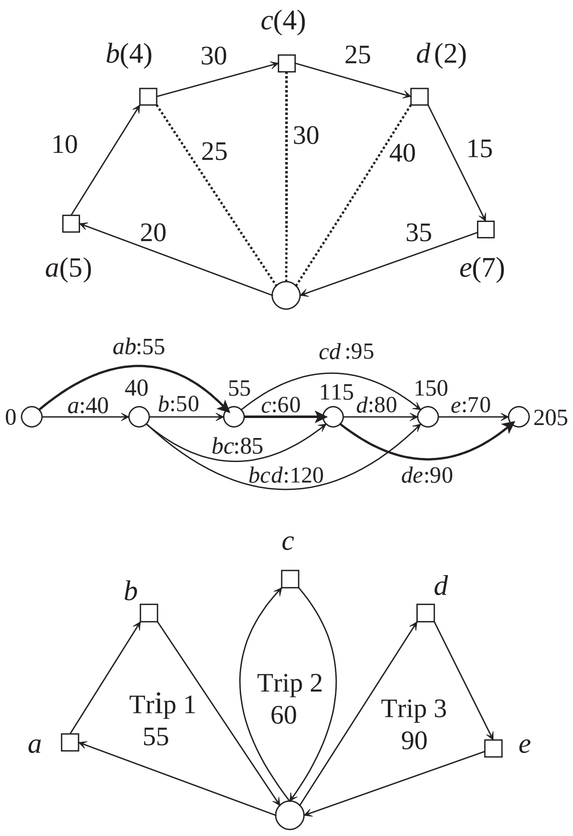

Fig. 1. Example of chromosome evaluation.

The vector of labels P is kept with the chromosome to extract the DVRP solution at the end of the GA, using the algorithm of Fig. 3. It builds up to n trips (worst case with one vehicle per demand). Each trip is a list of clients, possibly empty. The procedure *enqueue* adds a node at the end of a trip. The number of non-empty trips (or vehicles) actually used is given by t.

4.3. Extensions to other objective functions

Split is very flexible. The three examples hereafter show how to tackle more complicated objective functions. The $O(n^2)$ time complexity is maintained in the two first examples.

Total cost and number of vehicles. The algorithm of Fig. 2 may be adapted to break ties on cost with the number of required vehicles: (a) define for each node a second label N_i for the number of arcs on the shortest path, (b) set N_0 to 0 at the beginning, (c) in the case $V_{i-1} + cost < V_j$, add $N_j := N_{i-1} + 1$, (d) add a case $V_{i-1} + cost = V_j$ that updates P_j and N_j if $N_{i-1} + 1 < N_j$. It is also possible to minimize first the number of vehicles, by testing N_j before V_j.

Vehicle fleet mix problem (*VFMP*): In the VFMP, the fleet is not yet purchased. Several types of vehicles are possible, with given capacities and prices. The goal is to find a fleet composition

```
$V_0$ := 0
for $i$ := 1 to $n$ do $V_i$ := $+\infty$ endfor
for $i$ := 1 to $n$ do
   $load$ := 0; $cost$ := 0; $j$ := $i$
   repeat
      $load$ := $load$ + $q_{S_j}$
      if $i = j$ then
         $cost$ := $c_{0,S_j} + d_{S_j} + c_{S_j,0}$
      else
         $cost$ := $cost - c_{S_{j-1},0} + c_{S_{j-1},S_j} + d_{S_j} + c_{S_j,0}$
      endif
      if ($load \leq W$) and ($cost \leq L$) then
         //here substring $S_i \ldots S_j$ corresponds to arc $(i-1, j)$ in $H$
         if $V_{i-1} + cost < V_j$ then
            $V_j := V_{i-1} + cost$
            $P_j := i - 1$
         endif
         $j$ := $j + 1$
      endif
   until ($j > n$) or ($load > W$) or ($cost > L$)
enfor.
```

Fig. 2. Algorithm for the splitting procedure *split*.

```
for $i$ := 1 to $n$ do $trip(i)$ := $\emptyset$ endfor
$t$ := 0
$j$ := $n$
repeat
   $t$ := $t + 1$
   $i$ := $P_j$
   for $k$ := $i + 1$ to $j$ do $enqueue(trip(t), S_k)$ endfor
   $j$ := $i$
until $i = 0$.
```

Fig. 3. Algorithm to extract the VRP solution from vector *P*.

minimizing the operating cost (total cost of the trips) plus the fleet cost. To adapt the algorithm, when examining a trip $S_i \cdots S_j$, just add to *cost* the cost of the cheapest vehicle with a capacity not smaller than *load*.

Limited number of vehicles: The VFMP corresponds to a heterogeneous fleet with an unlimited number of vehicles for each type. It is also possible to handle a limited number of identical vehicles (homogeneous fleet) or a heterogeneous fleet with a limited number of vehicles for each type. The latter problem corresponds to a resource-constrained shortest path in H. Although it is NP-hard, it can be solved in pseudo-polynomial time [18].

```
Rank:  1   2   3   4   5   6   7   8   9
                  i=4     j=6
                   ↓       ↓
P1  :  1   3   2 | 6   4   5 | 9   7   8
P2  :  3   7   8 | 1   4   9 | 2   5   6

C1  :  8   1   9   6   4   5   2   3   7
C2  :  2   6   5   1   4   9   7   8   3
```

Fig. 4. Example of OX crossover.

5. Crossover

Our chromosomes without trip delimiters can undergo classical crossovers like the *order crossover* OX or the *linear order crossover* LOX. Traditionally, LOX is used when chromosomes clearly have two extremities (e.g., when they encode a Hamiltonian path or a one-machine schedule), while OX is better suited for cyclic permutations like in the TSP. Since one DVRP solution can give different chromosomes depending on the concatenation order of its trips, there is no special reason to distinguish a “first” or “last” trip. This is why we selected OX, after some testing confirming its superiority over LOX.

The example in Fig. 4 shows how OX constructs the first child $C1$. First, two cutting sites i and j are randomly selected in $P1$, here $i=4$ and $j=6$. Then, the substring $P1(i)\cdots P1(j)$ is copied into $C1(i)\cdots C1(j)$. Finally, $P2$ is swept circularly from $j+1$ onward to complete $C1$ with the missing nodes. $C1$ is also filled circularly from $j+1$. The other child $C2$ may be obtained by exchanging the roles of $P1$ and $P2$. The whole procedure can be implemented in $O(n)$.

6. Local search as mutation operator

To compete with TS or SA heuristics, the classical GA framework must be hybridized with some kind of improvement procedure, giving a *hybrid GA* or *memetic algorithm* [19]. For the DVRP, we quickly obtained much better results by replacing simple mutation operators (like moving or swapping some nodes) by a local search procedure LS. This does not lead to a premature convergence, thanks to the dispersal technique explained in Section 7.

A child C produced by OX is improved by LS with a fixed probability p_m. The sequence of nodes of C is first converted into a DVRP solution using algorithm 3. Then, LS scans the $O(n^2)$ neighborhoods listed below. Wider neighborhoods increase the GA running time without improving the final result. Small neighborhoods are sufficient because they are compensated by the intrinsic parallelism of the GA: indeed, the solutions which can be reached by crossovers from a given population define a kind of wide neighborhood that brings enough diversification.

Each iteration of LS scans in $O(n^2)$ all possible pairs of distinct nodes (u,v). These nodes may belong to the same trip or to different trips and one of them may be the depot. For each pair, the following simple moves are tested. x and y are the successors of u and v in their respective trips.

$T(u)$ is the trip visiting u.

M1. If u is a client node, remove u then insert it after v,
M2. If u and x are clients, remove them then insert (u,x) after v,
M3. If u and x are clients, remove them then insert (x,u) after v,
M4. If u and v are clients, swap u and v,
M5. If u, x and v are clients, swap (u,x) and v,
M6. If (u,x) and (v,y) are clients, swap (u,x) and (v,y),
M7. If $T(u)=T(v)$, replace (u,x) and (v,y) by (u,v) and (x,y),
M8. If $T(u)\neq T(v)$, replace (u,x) and (v,y) by (u,v) and (x,y),
M9. If $T(u)\neq T(v)$, replace (u,x) and (v,y) by (u,y) and (x,v).

Move 7 corresponds to the well-known 2-opt move, while moves 8–9 extend 2-opt to different trips. Each iteration of LS stops at the first improving move. The solution is then modified and the process repeated until no further saving can be found. Moves like M1 can empty a trip (which becomes a simple loop on the depot) but can also assign new clients to it later. This is why empty trips are removed only at the end of LS. The final solution with a cost $\lambda \leqslant F(C)$ is converted into a mutated chromosome M by concatenating its trips. *Split* is applied in all cases to M because it sometimes finds a better partition into trips for the same sequence (note that $F(M) \leqslant \lambda \leqslant F(C)$).

7. Population structure and initialization

The population is implemented as an array Π of σ chromosomes, always sorted in increasing order of cost to ease the basic GA iteration described in Section 8. So, the best solution is Π_1. To avoid useless recomputations, the fitness computed by *Split* is kept in each chromosome. This can be done by encoding each chromosome as a record with two fields, a sequence of nodes and a fitness value.

Clones (identical solutions) are forbidden in Π to ensure a better dispersal of solutions and to diminish the risk of premature convergence. This also allows a higher mutation rate p_m by local search, giving a more aggressive GA. To avoid comparing chromosomes in details and to speed-up clone detection, we impose a stricter condition: the costs of any two solutions must be spaced at least by a constant $\Delta > 0$, like in relation (6). We also tried an exact detection of clones, without observing a significant improvement. In the sequel, a population satisfying condition 6 will be said *well spaced.* The simplest form with $\Delta = 1$ (solutions with distinct integer costs) was already used in an efficient GA for the open-shop scheduling problem [15].

$$\forall P_1, P_2 \in \Pi : P_1 \neq P_2 \quad \Rightarrow \quad |F(P_1) - F(P_2)| \geqslant \Delta. \tag{6}$$

At the beginning, three good solutions are computed using the heuristics from Clarke and Wright [20] (parallel version in which each merger is followed by a 3-opt local search), Mole and Jameson [21], and Gillett and Miller [22]. These heuristics are denoted CW, MJ and GM in the sequel. The trips of each solution are concatenated into a chromosome. Like after a mutation by local search, *Split* must be applied to define the true fitness, which is sometimes a bit better than the cost found by the heuristic. The resulting chromosomes are stored in Π_1, Π_2 and Π_3.

Then, for $k = 4, 5, \ldots, \sigma$, each other chromosome Π_k is initialized as a random permutation of customers, evaluated by *Split*. Since a draw can yield a clone, we try up to τ times ($\tau = 50$ typically) to get an acceptable Π_k, i.e., such that the set $\Pi_1 \cdots \Pi_k$ is well spaced. If it is not possible to get the current chromosome Π_k, the population size is truncated to $\sigma = k - 1$ chromosomes. Hopefully, this happens only when n is very small (e.g., 10 customers) or when σ is too large. The population is finally sorted in increasing cost order.

Starting from an empty population, the initialization adds one chromosome at a time. Later on, in each GA iteration, only one chromosome is replaced (see Section 8). Therefore, the spacing may be checked at each step in $O(1)$ as follows. Note that two chromosomes P_1 and P_2 are well spaced iff their *scaled costs* $\lfloor F(P_1)/\Delta \rfloor$ and $\lfloor F(P_2)/\Delta \rfloor$ are different. Define a 0–1 vector U, indexed from 0 onward, such that $U(\varphi) = 1$ iff there exists in Π a solution with a scaled cost φ. If $F_{\max}$ is an upper bound on the fitness, U may be dimensioned up to index $\lfloor F_{\max}/\Delta \rfloor$. At the beginning, Π is empty and U is initialized to 0. A new chromosome K may be added to Π iff $U(\lfloor F(K)/\Delta \rfloor) = 0$. Once it is added, $U(\lfloor F(K)/\Delta \rfloor)$ must be set to 1.

8. GA Iterations and resulting general structure

8.1. Description of basic iteration

Parents are selected with the *binary tournament method*. Two chromosomes are randomly selected from the population and the least-cost one becomes the first parent $P1$. The procedure is repeated to get the second parent $P2$. OX is applied to generate two children $C1$ and $C2$. One child C is selected at random for replacing a mediocre chromosome Π_k drawn above the median, i.e., with $k \geqslant \lfloor \sigma/2 \rfloor$. This *incremental population management* preserves the best solution and allows a good child to reproduce quickly. If $(\Pi \setminus \{\Pi_k\}) \cup \{C\}$ is well spaced, the replacement is accepted: Π is re-sorted in $O(\sigma)$ by moving C to a sorted position, and one *productive iteration* (i.e., successful) is counted.

An excessive rejection rate (that is, most GA iterations are unproductive) appears if Δ or the population size σ are too large. This phenomenon worsens with a high mutation rate p_m because the local search LS compresses the population in a smaller cost interval. In practice, a reasonable rejection rate (at most 10% of iterations) is obtained with $\sigma \leqslant 50$, $0.2 \leqslant \Delta \leqslant 5$ and $p_m \leqslant 0.3$.

C is selected at random because the average GA performance is slightly inferior when Π_k is replaced by the best child or if two solutions are replaced by $C1$ and $C2$. When C undergoes mutation, the mutant is built in a separate chromosome M. If $(\Pi \setminus \{\Pi_k\}) \cup \{M\}$ is well spaced, M is assigned to the variable C. The goal of this trick is to try a replacement by C if its mutant is rejected (this is clear in the algorithm of Fig. 5). This also diminishes the rejection rate and slightly improves the final solution on average.

8.2. Main phase and restarts

The GA performs a main phase stopping after a maximum number of productive iterations, $\alpha_{\max}$ (number of crossovers that do not yield a clone), after a maximal number of iterations without improving the best solution, $\beta_{\max}$, or when it reaches the optimum cost or a lower bound (LB)

```
//initial heuristic solutions
$\Pi_1$ := solution of heuristic CW
$\Pi_2$ := solution of heuristic MJ
if not $spaced(\{\Pi_1, \Pi_2\})$ then $k$ := 1 else $k$ := 2 endif
$\Pi_{k+1}$ := solution of heuristic GM
if $spaced(\{\Pi_1 \dots \Pi_{k+1}\})$ then $k$ := $k$ + 1 endif
//initial random solutions
repeat
   $k$ := $k+1$
   $try$ := 0
   //try up to $\tau$ times to get $\Pi_k$
   repeat
      $try$ := $try+1$
      generate $\Pi_k$ at random
   until $spaced(\{\Pi_1 \dots \Pi_k\})$ or $(try > \tau)$
until $(k = \sigma)$ or $(try > \tau)$
if $try > \tau$ then $\sigma$ := $k-1$ endif
sort $\Pi$ in increasing cost order
//main loop of GA
$\alpha, \beta$ := 0
//iterations and iterations without improvement
repeat
   select two parents $P_1$ and $P_2$ by binary tournament
   apply OX to $(P_1, P_2)$ and select one child $C$ at random
   select $k$ at random in $[\lfloor \sigma/2 \rfloor, \sigma]$
   if $random < p_m$ then //mutation
      $M$ := $C$ mutated with the local search LS
      //if $M$ is a clone, replacement attempted with $C$
      if $spaced((\Pi \setminus \{\Pi_k\}) \cup \{M\})$ then $C$ := $M$ endif
   endif
   if $spaced((\Pi \setminus \{\Pi_k\}) \cup \{C\})$ then //productive iteration
      $\alpha$ := $\alpha$ + 1
      if $F(C) < F(\Pi_1)$ then $\beta$ := 0 else $\beta$ := $\beta + 1$ endif
      $\Pi_k$ := $C$
      shift $\Pi_k$ to re-sort $\Pi$
   endif
until $(\alpha = \alpha_{max})$ or $(\beta = \beta_{max})$ or $(F(\Pi_1) = LB)$.
```

Fig. 5. General algorithm for the GA.

known for some instances (in that case Π_1 is of course optimal). This main phase is summarized in Fig. 5, in which *spaced*(A) is a boolean function true iff the set of chromosomes A is well spaced.

Some hard instances are improved by a few restarts based on a *partial replacement procedure* proposed by Cheung et al. [23] in a GA for a facility location problem. This procedure must be adapted for well-spaced populations. Let ρ the number of chromosomes to be replaced, typically $\sigma/4$. Randomly generate a well-spaced population Ω of ρ chromosomes such that $\Pi \cup \{\Omega_k\}$ is well spaced for $k = 1,2,\dots\rho$. For $k = 1,2,\dots\rho$, if $F(\Omega_k) < F(\Pi_\sigma)$ (i.e., if the new chromosome is better than the worst solution in Π), replace Π_σ by Ω_k. If not, cross Ω_k with all chromosomes in $\Pi \cup \Omega$, keep the best child C obtained by these crossovers, and replace Π_σ by C if $F(C) < F(\Pi_{nc})$.

In practice, several populations Ω must be tried to get ρ successful replacements. We limit each restart to five attempts (even if less than ρ solutions are replaced), because the partial replacement procedure may be time consuming on large instances. Compared with a blind replacement, the partial replacement procedure preserves the best solution and cannot increase the worst cost. Numerical experiments also indicate a quicker descent of the GA.

9. Computational evaluation

9.1. Implementation and benchmarks

The GA is implemented in the Pascal-like language Delphi 5, on a Pentium-3 PC clocked at 1 GHz under the operating system Windows 98. The evaluation is performed on two sets of *Euclidean* benchmark problems, i.e., the vertices are defined by points in the plane and the cost for each edge (i,j) is the euclidean distance between i and j.

The first set contains 14 classical problems proposed by Christofides et al. [4] and can be downloaded from the OR Library [24]. These files have 50–199 customers. The only ones with a route-length restriction (DVRPs) are files 6–10, 13 and 14, all with non-zero service times. The others are pure VRPs. The solutions computed by various TS and SA algorithms are given by Gendreau et al. [5] and Golden et al. [6].

The second set contains 20 large-scale instances (200–483 customers) presented by Golden et al. [6], see [25] for an Internet address. Instances 1–8 are DVRPs, but with null service times. Golden et al. describe in their paper a deterministic annealing algorithm called *record-to-record travel* (*RTR*) and list the solution costs obtained by this method and by a TS heuristic designed by Xu and Kelly [26], called XK in the sequel.

Apart from our GA, the only algorithm which has been evaluated on both sets is a new promising method called *granular tabu search* (*GTS*) and proposed by Toth and Vigo [27]. It is based on the observation that the longest edges of a network are seldom included in optimal solutions. This method discards all edges whose length exceeds a granularity threshold, dynamically adjusted during the algorithm.

9.2. Common GA parameters

The key-ideas listed in Section 3 are essential for the good performance of the GA. We have conducted a long study showing a moderate (+) or strong (++) degradation of the average solution values if only one of the following change is tried:

C1. Population size $\sigma < 25$ or $\sigma > 50(+)$,
C2. No good initial solution in Π, only one, or more than $5(+)$,

C3. Local search LS applied to each initial solution of $\Pi(+)$,
C4. Clones allowed in the population (no test on spacing) $(++)$,
C5. Parent selection by a roulette method $(+)$,
C6. Mutation by a simple move or swap of nodes instead of LS $(++)$,
C7. Generational population management $(+)$,
C8. Replacement with the best child of OX or with two children $(+)$,
C9. Replacing the worst solution, not one above the median $(+)$.

If all key-ideas are kept, the GA performs best with small populations of $\sigma = 30$ solutions, $\Delta = 0.5$ and a relatively high mutation rate p_m (up to 0.20). σ and p_m are small enough to avoid losing too much time in unproductive crossovers. Each restart replaces $\rho = 8$ solutions. The exact value of p_m and the balance between the main phase and the restarts depend on the sets of instances and are specified in the corresponding sub-sections.

Following a discussion with Rafael Martí, from the University of Valencia (Spain), our GA obviously has some connections with *scatter search* [28]: small populations, high mutation rate, restarts corresponding to the short phases of scatter search. However, our algorithm generates children in a stochastic way and its performance decreases if local search is applied to each initial solution. This latter point is probably the most surprising. It does not follow the general memetic algorithm template proposed by Moscato [19].

9.3. Results on Christofides instances

The best standard setting of parameters include the common parameters $(\sigma = 30, \Delta = 0.5, \rho = 8)$, completed as follows. The GA performs the main phase with $p_m = 0.05$, ending after $\alpha_{\max} = 30\,000$ productive crossovers, $\beta_{\max} = 10\,000$ crossovers without improving the best solution, or when the optimum is reached (it is known only for problems 1 and 12). If the optimum is not reached, the GA is restarted for 10 short phases with $\alpha_{\max} = \beta_{\max} = 2000$ and p_m increased to 0.1. The results are presented in Table 1.

The three first columns give the problem number, the number of customers n and the best-known solution value (BKS) reported in [5,6]. The best solution cost $F(\Pi_1)$ in the initial population is shown in column 4, it is always obtained by one of the three heuristics CW, MJ and GM. Columns 5–8 concern the GA in which the standard parameters are applied to all instances. Columns 5 (GA 3000) and 6 give the solution cost and the CPU time used in seconds for 3000 crossovers (roughly one tenth of the main phase). Column 7 (GA) and 8 give the same information for the best solution found. Column 9 (Best GA) presents the best result we have found so far with various parameter settings. The last line gives the average deviations in % to the best-known solutions and the average CPU times.

When the GA starts, the average deviations of the heuristics CW, MJ and GM to the best-known solutions are, respectively, 6.81, 10.76 and 12.78%. Thanks to a compensation effect, the deviation becomes 4.77% for the best of three solutions. Using one setting, the GA stops with an average deviation of 0.23% and retrieves eight best-known solutions (boldface). Using various settings, it reaches 10 best-known solutions with a deviation reduced to 0.08%. Note that most best-known solutions are probably optimal: they remain stable since the last improvements published in 1995 for problems 5 and 10.

Table 1
GA results for Christofides instances

Pb	n	BKS	$F(\Pi_1)$	GA 3000	Time	GA	Time	Best GA
1	50	∗524.61	531.90	**524.61**	0.50	**524.61**	0.50	**524.61**
2	75	835.26	902.04	850.32	5.06	**835.26**	46.36	**835.26**
3	100	826.14	878.35	833.50	14.01	**826.14**	27.63	**826.14**
4	150	1028.42	1078.54	1049.24	42.79	1031.63	330.11	1030.46
5	199	1291.45	1383.42	1330.26	107.22	1300.23	1146.52	1296.39
6	50	555.43	584.06	**555.43**	3.40	**555.43**	0.66	**555.43**
7	75	909.68	956.23	921.42	9.72	912.30	85.18	**909.68**
8	100	865.94	906.22	866.87	18.45	**865.94**	22.41	**865.94**
9	150	1162.55	1280.90	1165.68	80.19	1164.25	434.90	**1162.55**
10	199	1395.85	1531.98	1434.52	177.91	1420.20	1609.87	1402.75
11	120	1042.11	1049.43	**1042.11**	23.57	**1042.11**	17.85	**1042.11**
12	100	∗819.56	824.42	**819.56**	2.70	**819.56**	2.70	**819.56**
13	120	1541.14	1585.97	1548.58	46.80	1542.97	626.54	1542.86
14	100	866.37	868.50	**866.37**	15.76	**866.37**	5.16	**866.37**
Average			+ 4.79	+ 0.90	39.15	+ 0.23	311.17	+ 0.08

Asterisks denote proven optima. Boldface indicate best-known solutions retrieved. Averages in columns 4, 5, 7 and 9 are deviations to best-known solutions in %. CPU times in seconds on a 1 GHz Pentium-3 PC.

The average GA duration is 5.2 min (up to 26.8 min), but our implementation is not specially optimized and running times could be improved. For instance, lists of closest neighbors could be used to avoid scanning each neighborhood completely. Anyway, the GA limited to 3000 crossovers is already a very good heuristic, with five best-known solutions retrieved (including the two optimal ones), an average deviation of 0.9% and a much better running time of 39 s. Note that this truncated GA retrieves 80% of the saving obtained by the complete GA, in 13% of its running time.

The comparison with TS algorithms surveyed in [5,6] is difficult, because the authors seldom give their best-known solutions for one standard setting of parameters, as recommended nowadays. Generally, they give best solutions obtained with different settings and do not mention corresponding CPU times.

Nevertheless, we try in Table 2 to compare our GA with 11 TS and 2 SA for the VRP. All references for these algorithms can be found in [5] or [6]. Our table compiles several tables from Golden et al., plus the results obtained by the GTS from Toth and Vigo. The criteria are the average distance in % to best-known solutions (column Adbks), the number of best-known solutions retrieved (Nbks), and the average computation time in minutes. The table, sorted in increasing order of distance to best solutions, shows very encouraging results: only one published TS out of 10 does better than the GA. Even with one single parameter setting, the GA would be at rank 4, with an average deviation of 0.23%.

Concerning CPU times, Osman uses a Vax 8600, Gendreau a 36 MHz Silicon Graphics (about 5.7 MFlops), Xu and Kelly an unspecified DEC Alpha (26–43 MFlops, about 5 times faster than the Silicon Graphics according to Gendreau) and Toth and Vigo a 200 MHz PC (15 MFlops). The 1 GHz PC running the GA has an estimated power of 75 MFlops. The last column of Table 2 gives

Table 2
Comparison of VRP metaheuristics on the 14 Christofides instances

Kind	Authors	Year	Adbks %	Nbks	Avg time	Scaled
TS	Taillard	1993	0.05	12	—	—
GA	Prins	2001	0.08	10	5.2	1.46
TS	Xu-Kelly (seven problems)	1996	0.10	5	103.0	11.00
TS	Gendreau et al.	1994	0.20	8	—	—
TS	Taillard	1992	0.39	6	—	—
TS	Rego and Roucairol	1996	0.55	6	—	—
GTS	Toth and Vigo	1998	0.55	4	3.5	0.20
TS	Gendreau et al.	1991	0.68	5	—	—
TS	Rego and Roucairol	1996	0.77	4	—	—
TS	Gendreau et al.	1994	0.86	5	46.8	1.00
GA	Prins (3000 Xovers)	2001	0.90	5	0.7	0.20
TS	Osman	1993	1.01	4	26.1	?
TS	Osman	1993	1.03	3	34.0	?
SA	Osman	1993	2.09	2	—	—

CPU times in minutes on various computers.

scaled times: if T_a is the duration and P_a the computer power for one algorithm a, its scaled time is $(T_a/T_g) \times (P_a/P_g)$, with g standing for Gendreau's TS heuristic. The remarkable speed of GTS must be underlined. The GA is 7 times slower than GTS but 7.5 times quicker than the TS heuristic from Xu and Kelly. The truncated version of the GA (3000 crossovers) is as fast as GTS and its solution costs are still honorable.

9.4. Results on large-scale instances

The common parameters for these instances include again $\sigma = 30$, $\Delta = 0.5$ and $\rho = 8$. Instead of a long main phase followed by shorter restarts (too time consuming on such files), the GA executes 10 phases with $\alpha_{\max} = \beta_{\max} = 2000$ crossovers and with a relatively high mutation rate $p_m = 0.2$. Table 3 shows the excellent results obtained.

Using the same parameters for all instances, the average deviation to best-known solutions is 5.82% when the GA starts, −0.20% with nine improvements after 2000 crossovers (one phase), and −0.78% with 11 improvements at the end. More precisely, there is a striking difference between DVRP instances (files 1–8) and the other files containing ordinary VRPs. All best-known solutions to DVRP instances are improved, even after 2000 crossovers, with a final deviation of −2.57%. Basic VRPs are improved 3 times, with an average final cost slightly above the best-known cost (+0.41%). The average price to pay for solving these large-scale problems is nearly one hour of CPU time.

Table 4 compares the solution costs obtained with one parameter setting by the GA and the algorithms RTR, XK and GTS mentioned in 9.1. As underlined by Toth and Vigo [27], the best results reported for XK are based on truncated distances. If real numbers are used, all DVRP solutions (files 1–8) found by XK and listed by Golden et al. [6] become infeasible, explaining why they are

Table 3
GA results for large-scale instances with one parameter setting

Pb	n	BKS	$F(\Pi_1)$	GA 2000	CPU	GA	CPU
1	240	5736.15	5935.59	**5662.43**	12.43	**5648.04**	32.42
2	320	8553.03	8648.33	**8488.15**	30.85	**8459.73**	77.92
3	400	11402.75	11756.34	**11038.62**	57.40	**11036.22**	120.83
4	480	14336.36	14867.99	**13743.74**	85.47	**13728.80**	187.60
5	200	6680.36	6931.51	**6460.98**	4.22	**6460.98**	1.04
6	280	8712.76	8915.53	**8412.90**	13.16	**8412.90**	9.97
7	360	10515.33	10842.41	**10268.37**	37.97	**10267.50**	39.05
8	440	12036.24	12352.05	**11872.23**	76.10	**11865.40**	88.30
9	255	587.09	645.73	598.59	8.34	596.89	14.32
10	323	746.56	815.04	758.49	18.80	751.41	36.58
11	399	932.68	1004.04	952.15	28.15	939.74	78.50
12	483	1136.05	1225.50	1152.88	55.98	1152.88	30.87
13	252	868.70	949.39	881.98	6.56	877.71	15.30
14	320	1096.18	1205.37	1106.65	13.55	**1089.93**	34.07
15	396	1363.34	1503.23	1392.04	27.45	1371.61	110.48
16	480	1650.42	1818.49	1680.67	50.58	1650.94	130.97
17	240	709.90	759.58	717.09	7.67	717.09	5.86
18	300	1014.82	1059.28	1020.47	15.00	1018.74	39.33
19	360	1389.14	1452.89	1404.09	25.67	**1385.60**	74.25
20	420	1875.17	1928.61	**1868.36**	41.57	**1846.55**	210.42
Average			5.82	− 0.20	30.84	− 0.78	66.60

Boldface indicate best-known solutions improved. Averages in columns 4, 5 and 7 are deviations to best-known solutions in %. CPU times in minutes on a 1 GHz Pentium-3 PC.

not included in the table. Moreover, the costs for files 17–20 increase (the table indicate corrected values). Note that double-precision real numbers are absolutely necessary to have two exact decimals at the end of any metaheuristic.

The GA gives clearly the best results, with 12 best solutions out of 20. The computation times are given in minutes for a 100 MHz Pentium PC at 8 MFlops (RTR), for a DEC Alpha at 26 MFlops (XK), for a 200 MHz Pentium PC at 15 MFlops (GTS), and for a 1 GHz Pentium PC at 75 MFlops (GA). The scaled times using RTR as reference level show that the GA is much slower than GTS and RTR but much faster than XK.

A few comments are necessary to have a fair comparison between GTS and the GA. GTS is designed to get an excellent compromise between solution cost and computation time. This objective is clearly achieved: GTS is by far the fastest metaheuristic for solving the VRP. Moreover, without the GA, GTS would have 12 best solutions in the table (problems 2, 3, 5 to 8, 13, 14, 16 to 18). The goal of our work is not a fast algorithm, but rather to bring the effective GA that was missing for the VRP and to define new best-known solutions, as a challenge for other metaheuristics.

The best solutions found by the GA and by GTS are often improved if several combinations of parameters are used. Column 3 of Table 5 presents the best-known solutions before the GA: 16 out of 20 are obtained by GTS (two of them are given with one decimal by Toth and Vigo). The GA

Table 4
Results of RTR, XK, GTS and GA on large-scale instances (one parameter setting)

Pb	n	RTR	CPU	XK	CPU	GTS	CPU	GA	CPU
1	240	5834.60	3.68	—	802.87	5736.15	4.98	**5648.04**	32.42
2	320	9002.26	22.66	—	898.53	8553.03	8.28	**8459.73**	77.92
3	400	11879.95	40.04	—	1749.27	11402.75	12.94	**11036.22**	120.83
4	480	14639.32	122.61	—	2432.42	14910.62	15.13	**13728.80**	187.60
5	200	6702.73	11.24	—	591.40	6697.53	2.38	**6460.98**	1.04
6	280	9016.93	18.79	—	913.70	8963.32	4.65	**8412.90**	9.97
7	360	11213.31	22.55	—	1062.73	10547.44	11.66	**10267.50**	39.05
8	440	12514.20	111.37	—	1586.20	12036.24	11.08	**1865.40**	88.30
9	255	**587.09**	23.01	589.10	340.20	593.35	11.67	596.89	14.32
10	323	749.15	31.49	**746.56**	501.82	751.66	15.83	751.41	36.58
11	399	934.33	69.19	**932.68**	852.72	936.04	33.12	939.74	78.50
12	483	**1137.18**	101.09	1140.72	1151.10	1147.14	42.90	1152.88	30.87
13	252	881.04	6.01	881.07	1465.77	**868.80**	11.43	877.71	15.30
14	320	1103.69	21.83	1118.09	1577.30	1096.18	14.51	**1089.93**	34.07
15	396	**1364.23**	32.62	1377.79	4340.07	1369.44	18.45	1371.61	110.48
16	480	1657.93	47.55	1656.66	8943.45	1652.32	23.07	**1650.94**	130.97
17	240	720.44	5.69	747.24	2314.00	**711.07**	14.29	717.09	5.86
18	300	1029.21	8.15	1066.57	4101.02	**1016.83**	21.45	1018.74	39.33
19	360	1403.05	12.42	1435.88	5718.38	1400.96	30.06	**1385.60**	74.25
20	420	1875.17	31.05	1934.99	10839.73	1915.83	43.05	**1846.55**	210.42
Nbest		3		2		3		12	
Time			37.15		2609.13		17.54		66.60
Scaled			1.00		228.24		0.88		16.80

Bold indicate best solutions among RTR, XK, GTS and GA. CPU times in minutes.

finds better solutions for all DVRP instances (files 1–8) and for four ordinary VRP instances. The average saving is 1.17%. Thirteen best-known solutions are now provided by the GA (boldface), four by GTS, two by XK and one by RTR.

10. Conclusion

This paper presents the first hybrid GA for the VRP able to compete with powerful TS algorithms in terms of average solution cost. On Christofides instances, this GA outperforms all metaheuristics published, except one. It becomes the best algorithm available for the large-scale instances generated by Golden et al. [6].

These very good results can be explained by some key-features. A possible premature convergence due to the local search is prevented by using small populations of distinct solutions. Three classical heuristics provide good starting points. The incremental population management and the partial replacement technique used in restarts accelerate the decrease of the objective function.

Table 5
New status of best-known solutions for large-scale instances

Pb	n	Previous best	Method	Best GA soln
1	240	5736.15	GTS	**5646.63**
2	320	8553.03	GTS	**8447.92**
3	400	11402.75	GTS	**11036.22**
4	480	14336.36	GTS	**13624.52**
5	200	6680.36	GTS	**6460.98**
6	280	8712.76	GTS	**8412.80**
7	360	10515.33	GTS	**10195.59**
8	440	12036.24	GTS	**11828.78**
9	255	587.09	RTR	591.54
10	323	746.56	XK	751.41
11	399	932.68	XK	933.04
12	483	1136.05	GTS	1133.79
13	252	868.7	GTS	875.16
14	320	1096.18	GTS	**1086.24**
15	396	1363.34	GTS	1367.37
16	480	1650.42	GTS	1650.94
17	240	709.9	GTS	710.42
18	300	1014.82	GTS	**1014.80**
19	360	1389.14	GTS	**1376.49**
20	420	1875.17	RTR	**1846.55**

The splitting algorithm that computes the fitness allows simple chromosomes and crossovers like for the TSP. Apart from these ideas, the GA remains relatively simple.

However, one point needs improvement: the GA is still slower than many TS algorithms, although it is faster than some of them. This drawback partly comes from the current implementation, which could be optimized, and from the local search, based on basic neighborhoods and absorbing typically 95% of CPU time. Therefore, our main goal is now to speed up the local search, without sacrifying its average solution cost. We also wish to develop a scatter search and compare it with the GA, to quantify the impact of randomization in terms of solution costs and CPU times.

References

[1] Cormen TH, Leiserson CE, Rivest RL. Introduction to algorithms. Cambridge, MA: MIT Press, 1990.

[2] Toth P, Vigo D. Exact solution of the vehicle routing problem. In: Crainic TG, Laporte G, editors. Fleet management and logistics. Dordrecht: Kluwer, 1998. p. 1–31.

[3] Laporte G, Gendreau M, Potvin JY, Semet F. Classical and modern heuristics for the vehicle routing problem. International Transactions in Operational Research 2000;7:285–300.

[4] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. New York: Wiley, 1979. p. 315–38.

[5] Gendreau M, Laporte G, Potvin J-Y. Metaheuristics for the vehicle routing problem. GERAD research report G-98-52, Montréal, Canada, 1998.

[6] Golden BL, Wasil EA, Kelly JP, Chao IM. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic TG, Laporte G, editors. Fleet management and logistics. Dordrecht: Kluwer, 1998. p. 33–56.
[7] Van Breedam A. An analysis of the effect of local improvement operators in GA and SA for the vehicle routing problem. RUCA working paper 96/14, University of Antwerp, Belgium, 1996.
[8] Goldberg DE, Lingle R. Alleles, loci and the traveling salesman problem. In: Grefenstette JJ. editor. Proceedings of the First International Conference on Genetic Algorithms, Hillsdale, NJ: Lawrence Erlbaum, 1985. p. 154–9.
[9] Schmitt LJ. An empirical study computational study of genetic algorithms to solve order problems: an emphasis on TSP and VRPTC. PhD dissertation, University of Memphis, 1994.
[10] Schmitt LJ. An evaluation of a genetic algorithmic approach to the VRP. Working paper, Department of Information Technology Management, Christian Brothers University, Memphis, 1995.
[11] Oliver IM, Smith DJ, Holland JRC. A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ. editor. Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum, 1987. p. 224–30.
[12] Potvin J-Y. Genetic algorithms for the traveling salesman problem. Annals of Operations Research 1996;63:339–70.
[13] Potvin J-Y, Bengio S. The vehicle routing problem with time windows—Part II: genetic search. INFORMS Journal on Computing 1996;8:165–72.
[14] Thangiah SR. Vehicle routing with time windows using genetic algorithms. Report SRU-CpSc-TR-93-23, Slippery Rock University, Slippery Rock, 1993.
[15] Prins C. Competitive genetic algorithms for the open-shop scheduling problem. Mathematical Methods of Operations Research 2000;52(3):389–411.
[16] Lacomme P, Prins C, Ramdane-Chérif W. A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In: Boers EJW et al., editors. Applications of evolutionary computing. Lecture Notes in Computer Science, vol. 2037. Berlin: Springer, 2001. p. 473–83.
[17] Beasley JE. Route-first cluster-second methods for vehicle routing. Omega 1983;11:403–8.
[18] Jaumard B, Semet F, Vovor T. A two-phase resource constrained shortest path algorithm for acyclic graphs. GERAD research report G-96-48, Montréal, Canada, 1996.
[19] Moscato P. Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. New York: McGraw-Hill, 1999. p. 219–34.
[20] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12:568–81.
[21] Mole RH, Jameson SR. A sequential route-building algorithm employing a generalized savings criterion. Operational Research Quaterly 1976;27:503–11.
[22] Gillett BE, Miller LR. A heuristic algorithm for the vehicle dispatch problem. Operations Research 1974;22:340–9.
[23] Cheung BKS, Langevin A, Villeneuve B. High performing evolutionary techniques for solving complex location problems in industrial system design. Journal of Intelligent Manufacturing 2001;12(5–6):455–66.
[24] The OR Library. http://mscmga.ms.ic.ac.uk/jeb/orlib.
[25] Large VRPs. http://www-bus.colorado.edu/publications/workingpapers/kelly.
[26] Xu J, Kelly J. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science 1996;30:379–93.
[27] Toth P, Vigo D. The granular tabu search and its application to the VRP. Research Report OR-98-9, DEIS, University of Bologna, 1998, To appear in INFORMS Journal of Computing.
[28] Glover F, Laguna M, Martí R. Scatter search. In: Ghosh A, Tsutsui S, editors. Advances in evolutionary computing: theory and applications. Berlin: Springer, 2002.

Christian Prins is a professor of operations research and industrial engineering at the University of Technology of Troyes (UTT), France. He serves as the director of the Department of Industrial Systems Engineering. His research interests are in logistics, scheduling and software engineering aspects in combinatorial optimization.