Classical and Modern Heuristics for the Vehicle Routing Problem

Gilbert Laporte

GERAD and École des Hautes Études Commerciales
3000 chemin de la Côte-Sainte-Catherine
Montréal Canada H3T 2A7
gilbert@crt.umontreal.ca

Michel Gendreau

Jean-Yves Potvin

Département d'informatique et recherche opérationnelle
and Centre de recherche sur les transports
Université de Montréal, Case postale 6128, Succursale "Centre-ville"
Montréal, Canada H3C 3J7
michelg@crt.umontreal.ca potvin@iro.umontreal.ca

Frédéric Semet

Université de Valenciennes et du Hainaut-Cambresis
LAMIH, BP 311 - Le Mont Houy, 59303 Valenciennes Cedex, France
frederic.semet@univ-valenciennes.fr

March, 1999
Revised: October, 1999

Les Cahiers du GERAD

G–99–21

Abstract

This article is a survey of heuristics for the *Vehicle Routing Problem.* It is divided into two parts: classical and modern heuristics. The first part contains well-known schemes such as the savings method, the sweep algorithm and various two-phase approaches. The second part is devoted to tabu search heuristics which have proved the most successful metaheuristic approach. Comparative computational results are presented.

Key words: Routing, search algorithms.

Résumé

Cet article passe en revue les heuristiques pour le problème de tournées de véhicules. Il se divise en deux parties: les heuristiques classiques et les heuristiques modernes. La première partie décrit des méthodes connues comme la méthode des gains, l'algorithme de balayage et diverses heuristiques à deux phases. La seconde partie décrit les heuristiques de recherche avec tabou qui constituent l'approche métaheuristique la plus prometteuse. On présente des résultats numériques comparatifs.

Mots-clefs: Tournées de véhicules, algorithmes de recherche locale.

Acknowledgments: This work was partly supported by the natural Sciences and Engineering Research Council of Canada (NSERC) under grants OPG0038816, OPG0039682, OPG0036662, and OPG0184123, and by the Quebec FCAR program under grant 99ER–0289. This support is gratefully acknowledged.

1 Introduction

The *Vehicle Routing problem* (VRP) is defined a graph $G = (V, E)$ where $V = \{0, \ldots, n\}$ is a vertex set. In this article, we restrict our attention to the undirected case, i.e., $E = \{(i, j) : i, j \in V, i < j\}$ represents an edge set. Vertex 0 is a depot while the remaining vertices are customers. With each vertex of $V \setminus \{0\}$ is associated a non-negative demand q_i and with edge (i, j) is associated a non-negative cost or length c_{ij}. (Since G is undirected, we use c_{ij} and c_{ji} interchangeably.) The VRP consists of designing m vehicle routes of least total cost, each starting and ending at the depot, such that each customer is visited exactly once, the total demand of any route does not exceed the vehicle capacity Q, the length of any route does not exceed a preset maximal route length L. In some versions of the problem, m is fixed *a priori*. In others, it is a decision variable.

Several families of heuristics have been proposed for the VRP. These can be broadly classified into two main classes: *classical heuristics* developed mostly between 1960 and 1990, and *metaheuristics* whose growth has occurred in the last decade. Most standard construction and improvement procedures in use today belong to the first class. These methods perform a relatively limited exploration of the search space and generally produce good quality solutions within modest computing times. Moreover, most of them can be easily extended to account for the diversity of constraints encountered in real-life contexts. Therefore, they are still widely used in commercial packages. In metaheuristics, the emphasis is on performing a deep exploration of the most promising regions of the solution space. These methods typically combine sophisticated neighbourhood search rules, memory structures, and recombinations of solutions. The quality of solutions produced by these methods is usually much higher than that obtained by classical heuristics, but the price to pay is increased computing time. Moreover, the procedures are usually context dependent and require finely tuned parameters which may make their extension to other situations difficult. In a sense, metaheuristics are no more than sophisticated improvement procedures and they can simply be viewed as natural enhancements of classical heuristics.

The purpose of this article is to review some of the most important families of heuristics for the VRP. Section 2 is devoted to classical heuristics. Our review of metaheuristics, presented in Section 3, is restricted to tabu search (TS) methods since these have been by for the most successful. The heuristics were tested on a set of fourteen benchmark instances described in Christofides *et al.* (1979). This survey is extracted and adapted from Chapters 5 and 6 of the Toth and Vigo (2000) book on the VRP.

2 Classical Heuristics

Two main techniques are used for constructing VRP solutions: merging existing routes using a *savings criterion*, and gradually assigning vertices to vehicle routes using an *insertion cost*.

2.1 Savings Algorithms

The Clarke and Wright (1964) savings algorithm is perhaps the most widely known heuristic for the VRP. It applies to problems for which the number of vehicles is a decision variable, and it works equally well for directed or undirected problems. A parallel and a sequential version of the algorithm are available. The algorithm works as follows.

Step 1 (Savings computation). Compute the *savings* $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ for $i, j = 1, \ldots, n$ and $i \neq j$. Create n vehicle routes $(0, i, 0)$ for $i = 1, \ldots, n$. Order the savings in a non-increasing fashion.

Parallel version

Step 2 (Best feasible merge). Starting from the top of the savings list, execute the following. Given a saving s_{ij}, determine whether there exist two routes, one starting with $(0, j)$, and the other one ending with $(i, 0)$, that can feasibly be merged. If so, combine these two routes by deleting $(0, j)$ and $(i, 0)$ and introducing (i, j).

Sequential version

Step 2 (Route extension). Consider in turn each route $(0, i, \ldots, j, 0)$. Determine the first saving s_{ki} or $s_{j\ell}$ that can feasibly be used to merge the current route with another route ending with $(k, 0)$ or starting with $(0, \ell)$. Implement the merge and repeat this operation to the current route. If no feasible merge exists, consider the next route and reapply the same operations. Stop when no route merge is feasible.

There is a great variability in the numerical results reported for the savings heuristics and authors often do not mention whether the parallel or the sequential version is considered. We have implemented both versions of the savings method combined or not with 3-opt exchanges (Lin, 1965). In 3-opt, an attempt to improve a vehicle route is made by repeatedly removing three of its edges, and reconnect the three resulting chains in all possible ways. This process stops when no further improvement is possible. Two variants of 3-opt have been tested. In the first one, FI, the first improving move is performed, whereas in the second one, BI, the whole neighbourhood is explored to identify the best improvement. Our results indicate that the parallel versions of the savings method clearly dominate the sequential ones. The best solutions are consistently obtained by the parallel savings heuristic combined with 3-opt and BI. This algorithm produces solutions whose value is on the average 6.71% above that of the best known.

Several enhancements to the Clarke and Wright algorithms have been proposed, namely by Gaskell (1967), Yellow (1970), Golden *et al.* (1977), Paessens (1988), and Nelson *et al.* (1985). Most are aimed at reducing computation time and memory requirements. However, given the power of present day computers, these improvements are less and less justified.

A number of authors have also attempted to optimize the merger of routes through the use of a matching algorithm (see, e.g., Desrochers and Verhoog (1989), Altinkemer and Gavish (1991), Wark and Holt (1994)). Applying a matching based algorithm yields improvements to the standard algorithm, but at the expense of much higher computation time.

2.2 Sequential Improvement Methods

Two well-known insertion methods have been proposed for the VRP. The first, due to Mole and Jameson (1976), expands one route at a time. The second, proposed by Christofides *et al.* (1979), applies in turn a sequential and a parallel route construction procedure. Both methods contain a 3-opt improvement phase. Computational results show that these algorithms are not competitive with the best available methods.

2.3 The Sweep Algorithm

The sweep algorithm applies to planar instances of the VRP. Feasible clusters are initially formed by rotating a ray centered at the depot. A vehicle route is then obtained for each cluster by solving a TSP. Some implementations include a post-optimization phase in which vertices are exchanged between adjacent clusters, and routes are reoptimized. To our knowledge, the first mentions of this type of method are found in a book by Wren (1971) and in a paper by Wren and Holliday (1972), but the sweep algorithm is commonly attributed to Gillett and Miller (1974) who popularized it. A simple implementation of this method is as follows. Assume each vertex i is represented by its polar coordinates (θ_i, ρ_i), where θ_i is the angle and ρ_i is the ray length. Assign a value $\theta_i^* = 0$ to an arbitrary vertex i^* and compute the remaining angles centered at 0 from the initial ray $(0, i^*)$. Rank the vertices in increasing order of their θ_i.

Step 1 (Route initialization). Choose an unused vehicle k.

Step 2 (Route construction). Starting from the unrouted vertex having the smallest angle, assign vertices to vehicle k as long as its capacity or the maximal route length is not exceeded. If unrouted vertices remain, go to Step 1.

Step 3 (Route optimization). Optimize each vehicle route separately by solving the corresponding TSP (exactly or approximately).

2.4 Petal Algorithms

A natural extension of the sweep algorithm is to generate several routes, called *petals*, and make a final selection by solving a set partitioning problem of the form

$$\text{Minimize} \quad \sum_{k \in S} d_k x_k$$

$$\text{subject to} \qquad \sum_{k \in S} a_{ik} x_k = 1 \qquad (i = 1, \ldots, n)$$

$$x_k = 0 \text{ or } 1 \qquad (k \in S),$$

where S is the set of routes, $x_k = 1$ if and only if route k belongs to the solution, a_{ik} is the binary parameter equal to 1 only if vertex i belongs to route k, and d_k is the cost of petal k. If routes correspond to contiguous sectors of vertices, then this problem possesses the column circular property and can be solved in polynomial time (Ryan, Hjorring and Glover, 1993).

This formulation was first proposed by Balinski and Quandt (1964), but becomes impractical when $|S|$ is large. Agarwal, Mathur and Salkin (1989) have used column generation to solve small instances of the VRP optimally ($10 \leq n \leq 25$). Heuristic rules for producing a promising subset S' of simple vehicle routes, called 1-petals, have been put forward by Foster and Ryan (1976) and by Ryan, Hjorring and Glover (1993). Renaud, Boctor and Laporte (1996b) go one step further by including in S' not only single vehicle routes, but also configurations, called 2-petals, consisting of two embedded or intersecting routes. The generation of 2-petals is quite involved and will not be described here.

Renaud, Boctor and Laporte (1996b) have compared their results with their own implementation of the sweep algorithm (Gillett and Miller, 1974) and of the petal algorithm of Foster and Ryan (1976). The 14 standard benchmark problems were solved with real distances. Results indicate that the 2-Petal algorithm produces solutions whose value is on the average 2.38% above that of the best known (compared with 7.09% for Sweep and 5.85% for 1-Petal). In comparison, our best implementation of the savings algorithm gave an average deviation of 6.71%. Average computing times are 1.76 seconds for Sweep, 0.26 second for 1-Petal and 3.48 seconds for 2-Petal. The larger times taken by Sweep and 2-Petal are due to the post-optimization phase which is absent from 1-Petal. Sweep uses 3-opt, whereas 2-Petal uses 4-opt* (Renaud, Boctor and Laporte, 1996a).

2.5 Cluster-First, Route-Second Algorithms

The Fisher and Jaikumar algorithm is probably the best known cluster-first, route-second algorithm. Instead of using a geometric method to form the clusters, it solves a Generalized Assignment Problem (GAP). It can be described as follows.

Step 1 (Seed selection). Choose seed points j_k in V to initialize each cluster k.

Step 2 (Allocation of customers to seeds). Compute the cost d_{ik} of allocating each customer i to each cluster k as $d_{ij_k} = \min\{c_{0i} + c_{ij_k} + c_{j_k0},\ c_{0j_k} + c_{j_ki} + c_{i0}\} - (c_{0j_k} + c_{j_k0})$.

Step 3 (Generalized assignment). Solve a GAP with costs d_{ij}, customer weights q_i and vehicle capacity Q.

Step 4 (TSP solution). Solve a TSP for each cluster corresponding to the GAP solution.

The number of vehicle routes m is fixed a priori in the Fisher and Jaikumar heuristic. The authors propose a geometric method based on the partition of the plane into m cones according to the customer weights. The seed points are dummy customers located along the rays bisecting the cones. Once the clusters have been determined, the TSPs are solved optimally using a constraint relaxation based approach (Miliotis, 1976). However, the Fisher and Jaikumar (1981) article does not specify how to handle distance restrictions.

Bramel and Simchi-Levi (1995) describe a two-phase heuristic in which the seeds are determined by solving a capacitated location problems and the remaining vertices are gradually included into their allotted route in a second stage. The authors suggest first locating m seeds, called concentrators, among the n customer locations so as to minimize the total distance of customers to their closest seed, while ensuring that the total demand assigned to any concentrator does not exceed Q. Vehicle routes are then constructed by inserting at each step the customer assigned to that route seed having the least insertion cost. The authors show that the algorithm is asymptotically optimal (i.e., its solution value tends to the optimum as n tends to infinity), but its empirical performance is not competitive with that of the best methods.

2.6 Improvement Heuristics

Improvement heuristics for the VRP operate on each vehicle route taken separately, or on several routes at a time. In the first case, any improvement heuristic for the TSP can be applied. In the second case, procedures that exploit the multi-route structure of the VRP can be developed.

Most improvement procedures for the TSP can be described in terms of Lin's (1965) λ-opt mechanism. Here, λ edges are removed from the tour and the λ remaining segments are reconnected in all possible ways. If any profitable reconnection (the first or the best) is identified, it is implemented. The procedure stops at a local minimum when no further improvements can be obtained. Checking the λ-optimality of a solution can be achieved in $O(n^\lambda)$ time. Several modifications to this basic scheme have been developed. Lin and Kernighan (1973) modify λ dynamically throughout the search. Or (1976) proposed the Or-opt method which consists of displacing strings of 3, 2, or 1 consecutive vertices to another location. This amounts to performing a restricted form of 3-opt interchanges. Checking Or-optimality requires $O(n^2)$ time. In the same spirit, Renaud, Boctor and Laporte (1996a) have developed a restricted version of the 4-opt algorithm, called 4-opt*, which attempts a subset of promising reconnections between a chain of at most w edges, and another chain of two edges. Checking whether a solution is 4-opt* requires $O(wn^2)$ operations. Johnson and McGeoch (1997) have performed a thorough empirical analysis of these and other improvement procedures for the TSP and have concluded that a careful implementation of the Lin-Kernighan scheme yields the best results on the average.

The three references by Thompson and Psaraftis (1993), Van Breedam (1994), and Kinderwater and Savelsbergh (1997) provide descriptions of multi-route edge exchanges for the VRP. These encompass a large number of edge exchange schemes used by several authors (see, e.g., Stewart and Golden (1984), Dror and Levy (1986), Salhi and Rand (1987), Fahrion and Wrede (1990), Potvin *et al.* (1992), Osman (1993), Taillard (1993), etc.). The Thompson and Psaraftis paper describes a general "b-cyclic, k-transfer" scheme in which a circular permutation of b routes is considered and k customers from each route are shifted to the next route of the cyclic permutation. The authors show that applying specific sequences of b-cyclic, k-transfer exchanges (with $b = 2$ or b variable, and $k = 1$ or 2) yields interesting results. Van Breedam classifies the improvement operations as "string cross", "string exchange", "string relocation", and "string mix", which can all be viewed as special cases of 2-cyclic exchanges, and provides a computational analysis on a restricted number of test problems. Kinderwater and Savelsbergh define similar operations and perform experiments mostly in the context of the VRP with time windows.

We now summarize Van Breedam's analysis. The four operations considered are:

1) *String Cross* (SC): Two strings (or chains) of vertices are exchanged by crossing two edges of two different routes.

2) *String Exchange* (SE): Two strings of at most k vertices are exchanged between two routes.

3) *String Relocation* (SR): A string of at most k vertices is moved from one route to another, typically with $k = 1$ or 2.

4) *String Mix* (SM): The best move between SE and SR is selected.

To evaluate these moves, Van Breedam considers two local improvement strategies. The first, FI, consists of implementing the first move that improves the objective function. The second, BI, evaluates all possible moves and implements the best one. Van Breedam then defines a set of parameters that can influence the behaviour of the local improvement procedure. These parameters are: the initial solution (poor, good), the string length (k) for moves of type SE, SR, SM ($k = 1$ or 2), the selection strategy (FI, BI), the evaluation procedure for a string length $k > 1$ (evaluate all possible string lengths between a pair of routes, increase k when a whole evaluation cycle has been completed without identifying an improvement move). To compare the various improvement heuristics, Van Breedam selects 15 tests problems among 420 instances. However, nine of these include either pick-up and deliveries constraints, or time-windows constraints, and are therefore not relevant within the context of this study. The remaining six instances contain capacity constraints where all customers have the same demand, so that the capacity constraint is exactly satisfied and only SC or SE moves can be performed. Therefore, the following conclusions should be interpreted with caution. The first observation made by Van Breedam is that it is better to initiate the search from a good solution than from a poor one, both in terms of final solution quality and of computing time. Also, the best solutions are obtained when SE moves are performed with a string length $k = 2$. However, using $k = 2$ is about twice as slow as using $k = 1$. Overall, SE moves appear to be the best. This is confirmed in a further comparison of local improvement, simulated annealing and tabu search heuristics using various types of moves. The local improvement heuristic with SE moves yields solution values that are 2.2% above the best known, compared with 4.7% for SC moves, but computing times are more than four times larger with SC moves.

2.7 Computational Comparison

In Table 1, we compare several of the best known classical VRP heuristics on the fourteen Christofides *et al.* (1979) benchmark instances. With the exception of the Fisher and Jaikumar (1981) algorithm, all results correspond to real distances. Good solutions are obtained quickly by means of the Clarke and Wright (1964) and sweep algorithms (Gillett and Miller, 1974). Better solutions can be obtained by means of a variety of algorithmic enhancements, but none of them is competitive with the best TS implementations described in Section 3.

3 Tabu Search Heuristics

Tabu search starts from an initial solution x_1 and moves at each iteration t from x_t to its best neighbour x_{t+1}, until a stopping criterion is satisfied. If $f(x)$ denotes the cost of x, then $f(x_{t+1})$ is not necessarily less than $f(x_t)$. To avoid cycling, solutions that

	n	Type [1]	Clarke and Wright[2]	Wark and Holt[3]	Sweep [4]		1-Petal algorithm [5]		2-Petal algorithm [6]		Fisher and Jaikumar[7]		Bramel and Simchi-Levi[8]		Best known solution value
1	50	C	578.56	**524.6**	531.90	*0.12*	531.90	*0.10*	**524.61**	*0.76*	524	*9.3*	**524.6**	*68*	**524.61**[9]
2	75	C	888.04	835.8	884.20	*0.17*	885.02	*0.07*	854.09	*0.52*	857	*12.0*	848.2	*406*	**835.26**[9]
3	100	C	878.70	830.7	846.34	*1.18*	836.34	*0.32*	830.40	*3.84*	833	*17.7*	832.9	*890*	**826.14**[9]
4	150	C	1128.24	1038.5	1075.38	*2.53*	1070.50	*0.41*	1054.62	*5.93*	1014	*33.6*	1088.6	*2552*	**1028.42**[9]
5	199	C	1386.84	1321.3	1396.05	*3.60*	1406.84	*0.41*	1354.23	*6.21*	1420	*40.1*	1461.2	*4142*	**1291.45**[9]
6	50	C,D	616.66	**555.4**	560.08	*0.16*	560.08	*0.09*	560.08	*0.56*	560	*15.2*	—	—	**555.43**[9]
7	75	C,D	974.79	911.8	965.51	*0.19*	968.89	*0.07*	922.75	*0.43*	916	*20.6*	—	—	**909.63**[9]
8	100	C,D	968.73	878.0	883.56	*1.47*	877.80	*0.25*	877.29	*2.91*	885	*52.2*	—	—	**865.94**[9]
9	150	C,D	1284.63	1176.5	1220.71	*3.00*	1220.20	*0.26*	1194.51	*3.58*	1230	*121.3*	—	—	**1162.55**[10]
10	199	C,D	1521.94	1418.3	1526.64	*4.91*	1515.95	*0.35*	1470.31	*5.19*	1518	*136.6*	—	—	**1395.85**[9]
11	120	C	1048.53	1043.4	1265.65	*3.52*	1252.84	*0.61*	1109.14	*11.70*	—	—	1051.5	*1303*	**1042.11**[9]
12	100	C	824.42	**819.6**	919.51	*0.64*	824.77	*0.21*	824.77	*2.11*	824	*6.4*	826.1	*400*	**819.56**[9]
13	120	C,D	1587.93	1548.3	1785.30	*2.24*	1773.69	*0.26*	1585.20	*3.31*	—	—	—	—	**1541.14**[10]
14	100	C,D	868.50	**866.4**	911.81	*0.85*	894.77	*0.17*	885.87	*1.69*	876	*6.3*	—	—	**866.37**[9]

(1) C: capacity restrictions, D: distance restrictions.
(2) Parallel savings + 3-opt and best improvement, implemented by Laporte and Semet (2000). All computing times are negligeable.
(3) Wark and Holt (1994). Best of five runs.
(4) Gillett and Miller (1974), implemented by Renaud *et al.* (1996).
(5) Foster and Ryan (1976), implemented by Renaud *et al.* (1996).
(6) Renaud *et al.* (1996). Computing times for the sweep, 1-petal and 2-petal heuristics are seconds on a Sun Sparcstation 2 (210.5Mips, 4.2 Mflops), with 32 megabytes of RAM.
(7) Fisher and Jaikumar (1981). The rounding rule for distances is unspecified. Computing times are seconds on a DEC-10
(8) Bramel and Simchi-Levi (1995). Computing times are seconds on an RS6000, Model 550.
(9) Taillard (1993).
(10) Rochat and Taillard (1995).

Table 1: Computational comparison of some classical heuristics.

were recently examined are forbidden, or *tabu*, for a number of iterations. To alleviate time and memory requirements, it is customary to record an attribute of tabu solutions rather than the solutions themselves. The basic TS mechanism can be enhanced by several computational features such as diversification and intensification strategies, as described by Hertz, Taillard and de Werra (1997), and by Glover and Laguna (1993, 1997), for example.

Over the last ten years or so, TS has been applied to the VRP by several authors. Some of the first TS algorithms did not yield impressive results, but subsequent implementations were much more successful. These include the work of Gendreau, Hertz and Laporte (1994), Taillard (1993), Xu and Kelly (1996), Rego and Roucairol (1996). In addition, Rochat and Taillard (1995) have introduced a useful and powerful concept, *adaptive memory*, which can be used to enhance any TS based algorithm.

3.1 Taburoute

The Taburoute algorithm of Gendreau, Hertz and Laporte (1994) is rather involved and contains several innovative features. The neighbourhood structure is defined by all solutions that can be reached from the current solution by removing a vertex from its current route, and inserting it into another route containing one of its p nearest neighbours using GENI, a *Generalized Insertion* procedure developed by Gendreau, Hertz and Laporte (1992) for the TSP. This may result in eliminating an existing route or in creating a new one. A second important feature of Taburoute is that the search process examines solutions that may be infeasible with respect to the capacity or maximum route length constraints. More precisely, the objective function contains two penalty terms, one measuring overcapacity, the other measuring overduration, each weighted by a self-adjusting parameter: every ten iterations, each parameter is divided by 2 if all 10 previous solutions were feasible, or multiplied by 2 if they were all infeasible. This way of proceeding produces a mix of feasible and infeasible solutions and lessens the likelihood of being trapped in a local minimum. At various points during the search process, Taburoute reoptimizes the route in which a vertex has just been inserted. This is achieved by using the US (*Unstringing & Stringing*) TSP post-optimization routine also developed by Gendreau, Hertz and Laporte (1992).

Taburoute does not actually use a tabu list, but random tabu tags. Whenever a vertex is moved from route r to route s at iteration t, its reinsertion into route r is forbidden until iteration $t+\theta$, where θ is an integer randomly drawn from the interval $[5, 10]$. Yet another feature of Taburoute is the use of a *diversification* strategy which consists of penalizing vertices that have been moved frequently in order to increase the probability of considering slow moving vertices. The objective function is artificially increased by adding to it a term proportional to the absolute frequency of movement of the vertex v currently being considered. Finally, Taburoute uses *false starts*. Initially, several solutions are generated and a limited search is carried out on each of them. The best identified solution is then selected as a starting point for the main search.

We now provide a short description of Taburoute. The readers are referred to the original article (Gendreau *et al.* 1994) for a detailed discussion of the parameter choices. In what follows, W is the set of vertices considered as candidates for reinsertion into another route at each iteration, $q \leq |W|$ is the number of these vertices for which a tentative reinsertion is actually made, and k is the number of consecutive iterations without improvement.

Step 1 (Initialization). Generate $\lceil \sqrt{n}/2 \rceil$ initial solutions and perform TS with $W = V \setminus \{0\}$, $q = 5m$ and $k = 50$. This value of q ensures that the probability of selecting one vertex from each route is at least 90%.

Step 2 (Solution improvement). Starting with the best solution observed in Step 1, perform TS with $W = V \setminus \{v_0\}$, $q = 5m$ and $k = 50n$.

Step 3 (Intensification). Starting with the best solution observed in Step 2, perform TS with $k = 50$. Here W is the set of the $\lfloor |V|/2 \rfloor$ vertices that have been most often moved in Steps 1 and 2, and $q = |W|$.

3.2 Taillard's Algorithm

The Taillard (1993) TS implementation contains some of the features of Taburoute, namely random tabu durations and diversification. It defines neighbourhood using the λ-interchange generation mechanism (Osman, 1993). Rather than executing the insertions with GENI, the algorithm uses standard insertions, thus enabling each insertion to be carried out in less time, and feasibility is always maintained. Every so often, individual routes are reoptimized using the optimization algorithm of Volgenant and Jonker (1983).

A novel feature of Taillard's algorithm is the decomposition of the main problems into subproblems. In planar problems, these subproblems are obtained by initially partitioning vertices into sectors centered at the depot, and into concentric regions within each sector. Each subproblem can be solved independently, but periodical moves of vertices to adjacent sectors are necessary. This makes sense when the depot is centered and vertices are uniformly distributed in the plane. For non-planar problems, and for planar problems not possessing these properties, the author suggests a different partitioning method based on the computation of shortest spanning arborescences rooted at the depot. This decomposition method is particularly well suited for parallel implementation as subproblems can then be distributed among the various processors. The combination of these strategies yields excellent computational results.

3.3 Xu and Kelly's Algorithm

With respect to the previous two TS algorithms, Xu and Kelly (1996) use a more sophisticated neighbourhood structure. They consider swaps of vertices between two routes, a global repositioning of some vertices into other routes, and local route improvements. The global repositioning strategy solves a network flow model to optimally relocate given numbers of vertices into different routes. Approximations are developed to compute the ejection and insertion costs, taking vehicle capacity into account. Route reoptimizations are performed by means of 3-opt exchanges (Lin, 1965) and a TS improvement routine. The algorithm is governed by several parameters which are dynamically adjusted through the search. A pool of best solutions is memorized and periodically used to reinitiate the search with new parameter values. Overall, this algorithm has produced several best known solutions on benchmark instances, but it is fair to say that it is not as effective as some other

TS implementations. It tends to require a substantial computational effort and properly tuning its many parameters can be problematic.

3.4 Rego and Roucairol's Algorithm

The main feature of the Rego and Roucairol (1996) TS algorithm is the use of ejection chains to move from one solution to the next. An ejection consists of moving a vertex to the position occupied by another vertex, thus creating a chain reaction of ℓ levels. For a given route orientation, denote by v_{i-1} the predecessor of v_i and by v_{i+1} its successor. An ℓ level ejection chain consists of replacing the triplets $(v^k_{i-1}, v^k_i, v^k_{i+1})$ $(k = 0, \dots, \ell)$ by the triplets $(v^k_{i-1}, v^{k-1}_i, v^k_{i+1})$ $(k = 1, \dots, \ell)$ and of relocating v^ℓ_i. A *legitimacy condition* is defined to ensure that the resulting solution remains feasible, i.e., that no arc appears more than once. At a general step of the algorithm, a number of vertices are considered as candidates for an ejection. As in Taburoute, infeasible intermediate solutions are considered. A parallel implementation of this procedure was developed. Again, this TS implementation yields good quality results, but does not measure up to the best known algorithms.

3.5 The Adaptive Memory Procedure of Rochat and Taillard

One of the most interesting developments to have occurred in the area of Tabu Search in recent years is the concept of *Adaptive Memory* developed by Rochat and Taillard (1995). It is mostly used in TS, but its applicability is not limited to this type of metaheuristic. An adaptive memory is a pool of good solutions that is dynamically updated throughout the search process. Periodically, some elements of these solutions are extracted from the pool and combined differently to produce new good solution. In the VRP, vehicle routes selected from several solutions will be used as a starting point. The extraction process gives a larger weight to those routes belonging to the best solutions. When selecting these routes, care must be taken to avoid including the same customer twice in a solution. This restriction means that the selection process will often terminate with a partial solution that will have to be completed using a construction heuristic. Rochat and Taillard have shown that the application of an adaptive memory procedure can enhance a search strategy. This has enabled them to obtain two new best solutions on the 14 standard VRP benchmark instances.

3.6 The Granular Tabu Search of Toth and Vigo

Granular Tabu Search (GTS) is yet another very promising concept. It was recently introduced by Toth and Vigo (1998) and has yielded excellent results on the VRP. The main idea behind GTS stems from the observation that the longer edges of a graph only have a small likelihood of belonging to an optimal solution. Therefore, by eliminating all edges whose length exceeds a *granularity threshold* several unpromising solutions will never be considered by the search process. Toth and Vigo suggest using $\nu = \beta\bar{c}$, where β is a *sparsification parameter* typically chosen in the interval $[1.0, 2.0]$, and $\bar{c}$ is the average edge length of a solution obtained by a fast heuristic. If $\beta \in [1.0, 2.0]$, then the percentage of remaining edges in the graph tends to be in the 10% – 20% range. In practice the value of β is dynamically adjusted whenever the incumbent has not improved for a set number of iterations, and periodically decreased to its initial value. Neighbour solutions are obtained by performing a limited number of edge exchanges within the same route or between two different routes. The authors propose a procedure capable of examining all potential exchanges in $O(|E(\nu)|)$ time, where $E(\nu) = \{(i,j) \in E : c_{ij} \leq \nu\} \cup I$, and I is a set of important edges such as those incident to the depot or belonging to high quality solutions. Toth and Vigo have implemented a version of GTS containing several of the features of Taburoute. As results presented in Table 2 show, GTS produces excellent solutions within very short computing times.

3.7 Computational Comparison

As rightly pointed out by Barr *et al.* (1995) and by Golden *et al.* (1998), properly testing metaheuristics is fraught with difficulties:

- These algorithms are usually governed by several user controlled parameters. Parameter setting should be done on a different set of instances from those used for the final tests.
- "Standard" results should be reported for one setting of the parameters. "Best" results corresponding to the best parameter values should be given alongside.
- It is difficult to interpret computing times in the case of parallel implementations.
- Comparable instance values should be used in all comparisons. For example, in the case of the VRP, different distance truncation rules yield different optima (see, Gendreau, Hertz and Laporte, 1994).

There is now a consensus among researchers that stricter testing practices must be enforced, but this has not always been the case in the past. The comparative results presented in Table 2 should be read with these remarks in mind. These show that TS generally yields very good solution values on the 14 benchmark instances. Two of the best known solution values are known to be optimal and the remaining ones are probably very close to being optimal.

4 Conclusion

Research on the development of heuristics for the VRP has made considerable progress since the first algorithms where proposed in the early sixties. The most powerful TS algorithms are now capable of solving medium size instances to optimality or near-optimality. It is now time to move to a new set of larger instances, including those recently developed by Golden *et al.* (1998). On the algorithmic side, time has probably come to concentrate on the development of faster, simpler (with fewer parameters) and more robust algorithms, even if this causes a small loss in solution quality. These attributes are essential if we want to see more of our algorithms implemented in commercial packages. The GTS algorithm of Toth and Vigo (1998) is a step in the right direction and should stimulate further research.

	n	Type [1]	Taillard f^*	Taburoute standard f^*	Taburoute Time [3]	Taburoute best f^*	Rochat – Taillard f^*	Xu – Kelly f^*	Xu – Kelly Time [4]	Rego – Roucairol f^*	Toth-Vigo (GTS) f^*	Toth-Vigo (GTS) Time [7]	Best known solution value
1	50	C	**524.61**	**524.61**	*6.0*	**524.61**		**524.61** [5,6]	*29.22* [5,6]	**524.61**	**524.61**	*0.81*	**524.61** [5,8,9,11,12,13]
2	75	C	**835.26**	835.77	*53.8*	835.32		**835.26** [5,6]	*48.80* [5,6]	835.32	838.60	*2.21*	**835.26** [5,8]
3	100	C	**826.14**	829.45	*18.4*	**826.14**		**826.14** [5,6]	*71.93* [5,6]	827.53	828.56	*2.39*	**826.14** [5,8,9]
4	150	C	**1028.42**	1036.16	*58.8*	1031.07		1029.56 [5,6]	*149.90* [5,6]	1044.35	1033.21	*4.51*	**1028.42** [8]
5	199	C	1298.79	1322.65	*90.9*	1311.35	**1291.45**	1298.58 [5,6]	*272.52* [5,6]	1334.55	1318.25	*7.50*	**1291.45** [10]
6	50	C,D	**555.43**	**555.43**	*13.5*	**555.43**		**555.43** [6]	*30.67* [6]	**555.43**	**555.43**	*0.86*	**555.43** [5,8,9,11,12]
7	75	C,D	**909.68**	913.23	*54.6*	**909.68**		965.62 [6]	*102.13* [6]	**909.68**	920.72	*2.75*	**909.68** [8,9,11]
8	100	C,D	**865.94**	**865.94**	*25.6*	**865.94**		881.38 [6]	*98.15* [6]	866.75	869.48	*2.90*	**865.94** [8,9]
9	150	C,D	**1162.55**	1177.76	*71.0*	1162.89		No solution	*168.08* [6]	1164.12	1173.12	*5.67*	**1162.55** [8]
10	199	C,D	1397.94	1418.51	*99.8*	1404.75	**1395.85**	1439.29 [6]	*368.37* [6]	1420.84	1435.74	*9.11*	**1395.85** [10]
11	120	C	**1042.11**	1073.47	*22.2*	**1042.11**		**1042.11** [5,6]	*91.23* [5,6]	**1042.11**	1042.87	*3.18*	**1042.11** [5,8,9,11]
12	100	C	**819.56**	**819.56**	*16.0*	**819.56**		**819.56** [5,6]	*56.61* [5,6]	**819.56**	**819.56**	*1.10*	**819.56** [5,8,9,11,12,14]
13	120	C,D	**1541.14**	1573.81	*59.2*	1545.93		1618.55 [6]	*201.75* [6]	1550.17	1545.51	*9.34*	**1541.14** [8]
14	150	C,D	**866.37**	**866.37**	*65.7*	**866.37**		915.24 [6]	*152.98* [6]	**866.37**	**866.37**	*1.41*	**866.37** [8,9,11,12]

(1) C: capacity restrictions, D: distance restrictions.
(2) Minutes on a VAX 8600.
(3) Minutes on a Silicon Graphics Workstation (36MHz, 5.7 Mflops).
(4) Minutes on a DEC ALPHA Workstation (DEC OSF/1 v 3.0).
(5) Xu and Kelly (1996).
(6) Golden *et al.* (1998).
(7) Minutes on a Petium 200 MHg PC (about three times faster than the Silicon Graphics Workstation used for Taburoute, and twice slower than the DEC ALPHA Workstation used by Xu and Kelly).
(8) Taillard (1993).
(9) Gendreau, Hertz and Laporte (1994).
(10) Rochat and Taillard (1995).
(11) Rego and Roucairol (1996), parallel implementation.
(12) Toth and Vigo (1998).
(13) Optimal solution (see Hadjiconstantinou, Christofides and Mingozzi, 1995).
(14) Optimal solution (see Golden *et al.*, 1998).

Table 2: Computational comparison of some TS heuristics.

References

Agarwal, Y. Mathur, K. and Salkin, H.M. (1989) A set-partitioning-based exact algorithm for the vehicle routing problem, *Networks 19*, 731–749.

Altinkemer, K. and Gavish, B. (1991) Parallel savings based heuristic for the delivery problem, *Operations Research 39*, 456–469.

Balinski, M.L. and Quandt, R.E. (1964) On an integer program for a delivery program for a delivery problem, *Operations Research 12*, 300–304.

Barr, R.S. Golden, B.L. Kelly, J.P. Resende, M.G.C. and Stewart Jr. W.R. (1995) Designing and reporting on computational experiments with heuristic methods, *Journal of Heuristics 1*, 9–32.

Bramel, J.B. and Simchi-Levi D. (1995) A location based heuristic for general routing problems, *Operations Research 43*, 649–660.

Christofides, N. Mingozzi, A. and Toth, P. (1979) The Vehicle Routing Problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (eds), *Combinatorial Optimization.* Wiley, Chichester.

Clarke, G. and Wright, J.W. (1964) Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research 12*, 568–581.

Desrochers, M. and Verhoog, T.W. (1989) A Matching Based Savings Algorithm for the Vehicle Routing Problem. Les Cahiers du GERAD G–89–04, École des Hautes Études Commerciales de Montréal.

Dror, M. and Levy, L. (1986) A vehicle routing improvement algorithm. — Comparison of a 'Greedy' and a 'Matching' implementation for inventory routing, *Computers & Operations Research 13*, 33–45.

Fahrion, R. and Wrede, W. (1990) On a principle of chain-exchange for vehicle-routing problems (1-VRP), *Journal of the Operational Research Society 41*, 821–827.

Fisher, M.L. and Jaikumar, R. (1981) A generalized assignment heuristic for vehicle routing, *Networks 11*, 109–124.

Foster, B.A. and Ryan, D.M. (1976) An integer programming approach to the vehicle scheduling problem, *Operations Research 27*, 367–384.

Gaskell, T.J. (1967) Bases for vehicle fleet scheduling, *Operational Research Quarterly 18*, 218.

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

Gendreau, M. Hertz, A. and Laporte, G. (1992) New insertion and postoptimization procedures for the traveling salesman problem, *Operations Research 40*, 1086–1094.

Gillett, B.E. and Miller, L.R. (1974) A heuristic algorithm for the vehicle dispatch problem, *Operations Research 22*, 240–349.

Glover, F. and Laguna, M. (1993) Tabu Search. In C.R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell, Oxford.

Glover, F. and Laguna, M. (1997) Tabu Search. Kluwer, Boston.

Golden, B.L. Magnanti, T.L. and Nguyen, H.Q. (1977) Implementing vehicle routing algorithms, *Networks 7*, 113–148.

Golden, B.L. Wasil, E.A. Kelly, J.P. and Chao, I-M. (1998) Metaheuristics in Vehicle Routing. In T.G. Crainic and G. Laporte (eds), *Fleet Management and Logistics.* Kluwer, Boston.

Hadjiconstantinou, E., Christophides, N. and Mingozzi, A. (1995) A New Exact Algorithm for the Vehicle Routing Problem Based on q-Paths and k-Shortest Path Relaxations. In M. Gendreau and G. Laporte (eds), *Freight Transportation*, *Annals of Operations Research 61*, 21–43.

Hertz, A. Taillard, É.D. and de Werra, D. (1997) Tabu Search. In E.H.L. Aarts and J.K. Lenstra (eds), *Local Search in Combinatorial Optimization.* Wiley, Chichester.

Johnson, D.S. and McGeoch, L.A. (1997) The Traveling Salesman Problem: A Case Study. In E.H.L. Aarts and J.K. Lenstra (eds), *Local Search in Combinatorial Optimization.* Wiley, Chichester.

Kinderwater, G.A.P. and Savelsbergh, M.W.P. (1997) Vehicle Routing: Handling Edge Exchanges. In E.H.L. Aarts and J.K. Lenstra (eds), *Local Search in Combinatorial Optimization.* Wiley, Chichester.

Laporte G. and Semet F. (2000) Classical Heuristics for the Vehicle Routing Problem. In P. Toth and D. Vigo (eds), *The Vehicle Routing Problem.*

Lin, S. (1965) Computer solutions of the traveling salesman problem, *Bell System Technical Journal 44*, 2245–2269.

Lin, S. and Kernighan, B. (1973) An effective heuristic algorithm for the traveling salesman problem, *Operations Research 21*, 498–516.

Miliotis, P. (1976) Integer programming approaches to the travelling salesman problem, *Mathematical Programming 10*, 367–378.

Mole, R.H. and Jameson, S.R. (1976) A sequential route-building algorithm employing a generalized savings criterion, *Operational Research Quarterly 27*, 503–511.

Nelson, M.D. Nygard, K.E. Griffin, J.H. and Shreve, W.E. (1985) Implementation techniques for the vehicle routing problem, *Computers & Operations Research 12*, 273–283.

Or, I. (1976) Traveling Salesman-Type Combinatorial Optimization Problems and their Relation to the Logistics of Regional Blood Banking, Ph.D. dissertation, Northwestern University, Evanston, IL.

Osman, I.H. (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Annals of Operations Research 41*, 421–451.

Paessens, H. (1988) The savings algorithm for the vehicle routing problem, *European Journal of Operational Research 34*, 336–344.

Potvin, J.-Y. Kervahut, T. Garcia, B.L. and Rousseau, J.-M. (1992) The vehicle routing problem with time windows — Part I: Tabu Search, *INFORMS Journal on Computing 8*, 158–164.

Rego, C. and Roucairol, C. (1996) A Parallel Tabu Search Algorithm using Ejection Chains for the Vehicle Routing Problem. In I.H. Osman and J.P. Kelly (eds), *Meta-Heuristics: Theory and Applications*. Kluwer, Boston.

Renaud, J. Boctor, F.F. and Laporte, G. (1996a) A fast composite heuristic for the symmetric traveling salesman problem, *INFORMS Journal on Computing 8*, 134–143.

Renaud, J. Boctor, F.F. and Laporte, G. (1996b) An improved petal heuristic for the vehicle routing problem, *Journal of the Operational Research Society 47*, 329–336.

Rochat, Y. and Taillard, É.D. (1995) Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics 1*, 147–167.

Ryan, D.M. Hjorring, C. and Glover, F. (1993) Extensions of the petal method for vehicle routing, *Journal of the Operational Research Society 44*, 289–296.

Salhi, S. and Rand, G.K. (1987) Improvements to vehicle routing heuristics, *Journal of the Operational Research Society 38*, 293–295.

Stewart Jr. W.R. and Golden, B.L. (1984) A Lagrangean relaxation heuristic for vehicle routing, *European Journal of Operational Research 15*, 84–88.

Taillard, É.D. (1993) Parallel iterative search methods for vehicle routing problems, *Networks 23*, 661–673.

Thompson, P.M. and Psaraftis, H.N. (1993) Cyclic transfer algorithms for the multivehicle routing and scheduling problems, *Operations Research 41*, 935–946.

Toth, P. and Vigo, D. (1998) Granular Tabu Search. Working paper, DEIS, University of Bologna.

Toth, P. and Vigo, D. (2000) The Vehicle Routing Problem, *Monograph on Discrete Mathematics and Applications, SIAM.*

Van Breedam, A. (1994) An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle-Related, Customer-Related, and Time-Related Constraints. Ph.D. dissertation, University of Antwerp.

Volgenant, A. and Jonker, R. (1983) The symmetric traveling salesman problem and edge exchange in minimal 1-trees, *European Journal of Operational Research 12*, 394–403.

Wark, P. and Holt, J. (1994) A repeated matching heuristic for the vehicle routing problem, *Journal of the Operational Research Society 45*, 1156–1167.

Wren, A. (1971) Computers in Transport Planning and Operation. Ian Allan, London.

Wren, A. and Holliday, A. (1972) Computer scheduling of vehicles form one or more depots to a number of delivery points, *Operational Research Quarterly 23*, 333–344.

Xu, J. and Kelly, J.P. (1996) A network flow-based tabu search heuristic for the vehicle routing problem, *Transportation Science 30*, 379–393.

Yellow, P. (1970) A computational modification to the savings method of vehicle scheduling, *Operational Research Quarterly 21*, 281–283.