Parallel Iterative Search Methods for Vehicle Routing Problems

É. Taillard

École Polytechnique Fédérale de Lausanne, Département de Mathématiques, CH-1015 Lausanne, Switzerland

This paper presents two partition methods that speed up iterative search methods applied to vehicle routing problems including a large number of vehicles. Indeed, using a simple implementation of taboo search as an iterative search method, every best-known solution to classical problems was found. The first partition method (based on a partition into polar regions) is appropriate for Euclidean problems whose cities are regularly distributed around a central depot. The second partition method is suitable for any problem and is based on the arborescence built from the shortest paths from any city to the depot. Finally, solutions that are believed to be optimum are given for problems generated on a grid. *© 1993 by John Wiley & Sons, Inc.*

1. INTRODUCTION

Vehicle routing problems (VRP) are among the (NP-hard) combinatorial optimization problems for which exact and heuristic methods have most evolved these last years. This is certainly due to the increase in the demand by transportation companies for methods to address this important problem area.

On the other hand, iterative local search techniques for combinatorial optimization have been more and more used for solving practical problems. Indeed, these techniques lead to methods that are well adapted for relatively small VRP (less than 150 cities): They are able to find very good solutions in a few seconds; with a few minutes of computation, they generally find solutions that are better than most of the other heuristic methods. For example, Gendreau et al. [6] showed that their adaptation of taboo search provides better solutions than do more than a dozen previously proposed methods. The aim of this paper was precisely to show that it is possible to adapt such iterative search methods for the treatment of larger problems.

The basic problem that we have studied is the following: Identical vehicles (the number of which is undefined), having a fixed carrying capacity Q, have to deliver order quantities q_i ($i = 1 \ldots n$) of goods to n cities from a single depot. Knowing the distance d_{ij} between cities i and j ($i, j = 0 \ldots n$, city 0 is the depot), the problem is to find tours for the vehicles in such a way that

- The total distance traveled by the vehicles is minimized (a tour starts from the depot and ends at the depot).
- Every city receives its delivery by a unique vehicle.
- The total quantity of goods that a single vehicle delivers cannot be bigger than Q.

We will also consider a variant of this problem for which the delivery at each city requires a service time and for which the time needed to perform any tour is limited by a value L; in this case, the distance d_{ij} is interpreted as a time.

Iterative search methods are generally easy to implement and they often produce very good solutions to the VRP. Moreover, they may easily be adapted to practical problems dealing with a large number of constraints (see, e.g., [16]). Among these iterative search

CCC 0028-3045/93/080661-13

methods, the taboo search (TS) techniques provide very good solutions, if independent adaptations due to [6] and [14] are considered. It seems likely that TS might find optimum solutions of small problems (50–100 cities). However, for larger problems, these adaptations are less and less efficient.

In this paper, we are interested in methods that are able to produce solutions of very high quality. It is shown in [6] that of other approaches to the VRP [1, 3, 4, 7, 13] the two-phase and the incomplete tree search algorithms of [2, 5, 9, 15, 19, 21] are not competitive with respect to the quality criterion: None of them has produced solutions less than 2% above the best-known solution, on average, for the set of 14 problems proposed by [2]. Some of these methods produce solutions more than 10% above the best-known solution, on average, and in some instances, the solutions produced may be nearly 30% (!) above the best-known solution.

This paper presents first a simple adaptation of TS to VRP in Section 2. Then, in Section 3, decomposition methods based on the coordinates of the cities are presented. These decompositions provide very good results for problems for which the cities are more or less uniformly distributed around a central depot. When the cities are very irregularly distributed, or when the distances between cities are not Euclidean, it is necessary to decompose the problem in another way. Such decomposition methods are presented in Section 4. The last section concludes with improvements that might be made to our methods.

2. ITERATIVE SEARCH FOR VRP

In a general way, an iterative search may be sketched as follows:

(a) Choose an initial solution of the problem s_0. Set $k = 0$.
(b) Whereas a stopping criterion is not met, repeat the following step:
    (c) Choose from a set $N(s_k)$ of neighbor solutions of s_k the next visited solution s_{k+1}.
    Set $k = k + 1$.

The choice of a policy for points (a)–(c) leads to various iterative searches. Generally, the initial solution is chosen in such a way that its generation is easy and fast. For VRP, a separate vehicle can be assigned to each city with each vehicle performing just the tour depot–city–depot.

The choice of the stopping criterion is often related to the type of iterative search chosen: A descent method stops as soon as there is no better solution than s_k in the neighborhood $N(s_k)$; a simulated annealing (SA) process stops when the number of steps performed without improving the best solution found up to now is greater than a threshold (this threshold is modified during the search, according to other parameters such as the temperature); TS stops when the number k of iterations performed is greater than a threshold (this threshold may vary during the search). Naturally, the stopping criteria listed here are not exhaustive and some authors have adopted more subtle criteria.

A very general neighborhood for the VRP is constituted by the following type of move: Having chosen two different tours A and B, every possible exchange of μ cities of tour A and π cities of tour B is tried [with $0 \le \mu \le \min(M, |A|)$, $0 \le \pi \le \min(P, |B|)$, where M and P are nonnegative integers]. We shall use the notation (M, P) to represent this type of move. The number of possible exchanges grows very fast with M and P, so iterative search methods use very small values of M and P, e.g., (1, 0) or (1, 1), that correspond, respectively, to the move of a city from tour A into tour B and to the exchange of two cities between tours A and B.

Osman [14] used a neighborhood of type (2, 2) for a descent method and a neighborhood of type (1, 1) for TS and SA. He concluded that TS is superior for such neighborhoods. Gendreau et al. [6] and Semet and Taillard [16] used a neighborhood of type (1, 0) for their implementations of TS. However, we suspect that better neighborhoods might exist, e.g., those of type (M, P) with $M, P \ge 2$ restricted to the exchange of consecutive cities on the tour.

As all the best-known solutions of a set of 14 problems proposed in [2] were found using TS [6], we choose to solve by this technique the subproblems that result from the decomposition of large problems. We shall briefly present the simple implementation of TS that we have designed for the VRP (for additional features on TS, as applied in a variety of combinatorial optimization settings, the reader may refer to [8]).

The initial solution that we chose is to use a separate vehicle for each city. The neighborhood type is (1, 1) with restriction to feasible solutions (i.e., two successive solutions are feasible and differ only by the exchange of two cities not belonging to the same tour or by the move of a city from one tour to another). The basic idea of TS is to choose for s_{k+1} the best solution belonging to $N(s_k)$. However, to avoid cyclically visiting the same solutions, the reverse of a move that has been performed is forbidden during a number t of iterations: If at iteration k city c_1, belonging to tour A, and city c_2, belonging to tour B, are exchanged (respectively, if city c is moved from tour A to tour B), it is forbidden to put both c_1 in A and c_2 in B (respectively: to put c in A) again during iterations

$k + 1$ to $k + t$, unless this move leads to a solution better than the best found by the search so far. This type of taboo restriction was also chosen by [14].

If TS is implemented only with this taboo list, it turns out that the cities that are near to the depot are much more frequently moved than are the others. This is due to the TS process that chooses the best-allowed neighbor at each step. To diversify the search, we penalize the moves that are frequently performed: A move m that is performed with a frequency f_m is penalized by a value $W \cdot f_m$, unless it improves the best solution found so far (W is a parameter whose best value depends upon the particular VRP). Such a diversification technique was successfully applied to other problems; see, e.g., to scheduling [18].

The values of the parameters t and W have to be adjusted so that the search produces good solutions. We randomly chose, uniformly between $0.4n$ and $0.6n$, the value of the parameter t. Let $\Delta_k^{\max}$ be the maximal absolute value of a feasible move tried up to iteration k (i.e., $\Delta_k^{\max} = \max_{j \le k, s \in N(s_j)} |f(s_j) - f(s)|$), and ν, the number of vehicles used in the best solution found up to iteration k; W (for iteration k) is randomly chosen, uniformly between $0.1 \cdot \Delta_k^{\max} \sqrt{n \cdot \nu}$ and $0.5 \cdot \Delta_k^{\max} \sqrt{n \cdot \nu}$. Intuitively, this formula is justified as follows: The penalty should be proportional to the value of the moves (this explains the factor $\Delta_k^{\max}$); since the number of moves increases with the size of the problem, it follows that the frequency at which each move is performed decreases as the size of the problem grows. The factor $\sqrt{n \cdot \nu}$ normalizes the decrease of the frequencies. We have observed that factors between 0.1 and 0.5 provide good results, whatever the problem type and size are.

To evaluate the cost of a move, two traveling salesman problems have to be solved: one for each modified tour. It is not reasonable to consider exact methods for solving these problems at each trial move or even at each iteration for the current solution because they are too time-consuming. So, we have to use a heuristic method for evaluating the value of moves. We have chosen the simplest insertion heuristic method: If city c is moved from tour A to tour B, one goes directly in tour A from the city that was preceding c to the city that was succeeding to c. In tour B, one inserts c at the place that least increases the length of tour B (in case of exchange moves, both cities are first removed and then inserted). Gendreau et al. [6] used a more elaborate insertion procedure, but if the tours contain a relatively small number of cities, it turns out that the possible lower quality of our insertion procedure is compensated by its much faster execution. If the problem is "regular," i.e., if every tour contains about n/ν cities, when n is the total number of cities and ν is the number of vehicles in a good solution, then it is possible to perform one iteration of our TS in $O(n\nu + n^2 + (n^3/\nu^2))$: One move may be evaluated in $O(n/\nu)$; as only two tours change from one iteration to the next one, only $O(n(n/\nu))$ moves have to be reevaluated (the values of the remaining ones have not changed and may be stored). Then, we have to choose the best among $O(n\nu + n^2)$ moves [because there are $O(n^2)$ that exchange two cities belonging to different tours and $O(n\nu)$ moves that put one city in another tour]. Generally, for a given type of problem, ν is proportional to n; in this case, the complexity of one step of tour TS is $O(n^2)$. For example, for a problem involving 50 cities and six vehicles, one iteration of our TS takes about 13 ms on a Silicon Graphics 4D/35 workstation.

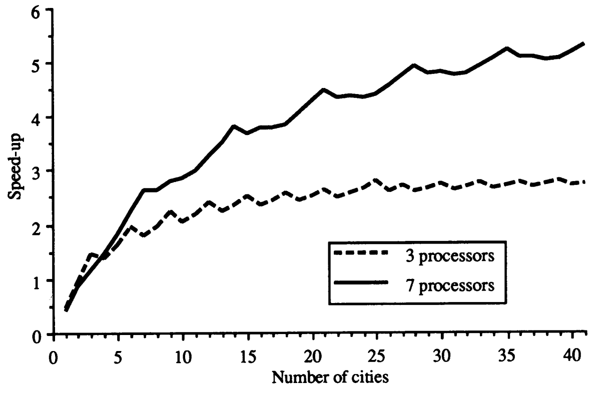

Fig. 1. Efficiency of the parallelization of the insertion method.

The insertion method may be easily implemented on distributed computers and interesting speed-ups may be obtained with small sizes of problems. Figure 1 shows the speed-ups that we have obtained experimentally as a function of the problem size with a network of three or seven transputers connected in binary tree. We see in this figure that this insertion procedure may be parallelized efficiently.

As this simple insertion procedure does not produce optimal tours in general, the tours may become rather poor. Thus, every 200 iterations, or when a solution is found at less than 0.1% above the best found so far during the search, every tour is optimized with the code of [20] for solving exactly traveling salesman problems. In such conditions, for the problems considered in this paper, the time spent in the exact computation of the tours is only a small percentage of the total CPU time and the quality of the solutions that our algorithm produces is much higher. Using an exact computation of the tours may be problematic if the number of cities per tour is greater than 40 or 50; this problem has never occurred for the problem instances considered in this paper.

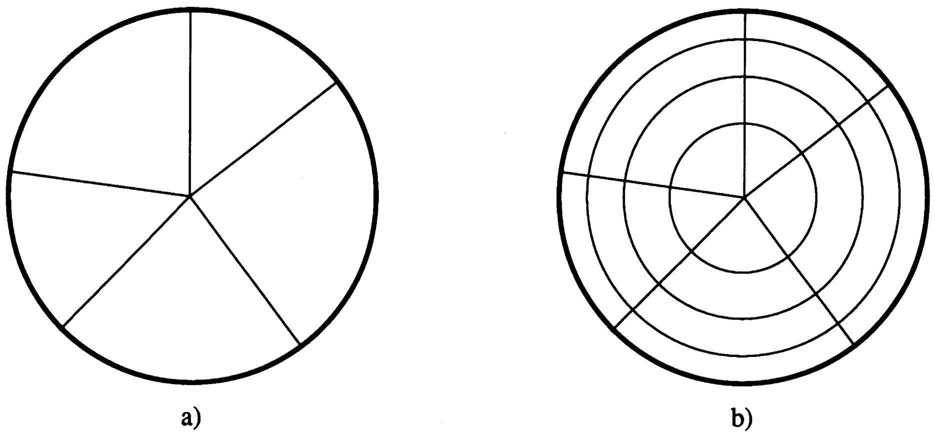

Fig. 2. Partition into sectors and partition into polar regions.

3. DECOMPOSITION OF UNIFORM PROBLEMS

By "uniform problems," we mean problems for which the depot is almost centered and for which the cities are regularly distributed around the depot, without forming distinct clusters. Almost all the problems proposed by [2] are "uniform," but the problem with 120 cities has an uncentered depot and the cities are clearly forming groups. We shall see that the partition method proposed in this section works poorly (on average) for this latter problem.

When examining good solutions to problems that involve more than 100 cities and 10 vehicles (see, e.g., Fig. 5), it turns out that locality properties for iterative searches exist: It is useless to try to move a city to every other tour; it is sufficient to try to move this city to some of the nearest tours. Moreover, a move modifies only two tours. So it is possible to simultaneously perform another move that involves cities belonging to other tours. If the problem under consideration has cities uniformly distributed around the depot, a partition of the problem into sectors [see Fig. 2(a)] is recommended. For very large problems, a partition of the sectors themselves may be necessary. Thus, we have a partition into polar regions [see Fig. 2(b)]. Such partitions were proposed in [12] for a slightly different version of the problem. In this reference, heuristic methods using such partitions are shown to satisfy, asymptotically, the "strongest possible optimality property." For the problem that we are dealing with, there is a theoretical background justifying the decomposition of large problems: An interpretation of the results of [17] is that, when the number of cities tends to infinity, problems randomly generated (under certain assumptions) may be partitioned into subproblems so that the value of the optimum solution of the full problem is equal, asymptotically, to the sum of the values of the optimum solutions of the subproblems. The optimum solution value of problems randomly generated tends to a value also given in [17]. Unfortunately, this value is of little use for the problems considered in this paper since for some problems it is less than 50% of the best-known solution value (and probably of the unknown optimum solution value as well); even for a problem with 1024 cities generated on a grid (the biggest problem considered in this paper), the value of [17] is still only 92% of the best-known solution value.

We adopt the following partition algorithm:

First partition of uniform problems:

Data: n cities of polar coordinates (θ_i, ρ_i) [$i = 1 \dots n$, depot at $(0, 0)$].
s : number of sectors.
c : number of partitions of the sectors.

(a) Renumber the cities by increasing angle θ_i (from now on, we have $\theta_i \le \theta_{i+1}$, $i = 1 \dots n - 1$).
(b) Generate a random integer g from 1 to n inclusive.
(c) Assign to sector j cities $((\lfloor (n/s)\cdot(j - 1) + 1.5\rfloor + g)\ mod(n)) + 1$ to $((\lfloor (n/s)\cdot j + 0.5\rfloor + g)\ mod(n)) + 1$.
(d) Renumber the cities assigned to every sector by increasing radius ρ_i; for sector j that contains n_j cities, we have now $\rho_i \le \rho_{i+1}$ $(1 \le i \le n_j - 1)$.
(e) Assign to region k of sector j cities $\lfloor (n_j/c)\cdot(k - 1) + 1.5\rfloor$ to $\lfloor (n_j/c)\cdot k + 0.5\rfloor$.

If the quantities to deliver to each city are not regularly distributed, a partition that assigns approximately the same delivery quantity to each region might be better than this partition that assigns approximately the same number of cities to each region.

Each region is then treated as an independent VRP and solved by a TS process. The solutions produced by this method are not good because they involve a number of vehicles that is too high. It is for this reason that we propose to evaluate (manually) the total number of vehicles required to deliver to every city and to distribute this number of vehicles among the regions. Every subproblem may construct a number of tours up to the number of vehicles assigned to this region. If a delivery is not made to a city, a cost of nondelivery equal to double the distance to the depot is assigned. Now, the solutions produced are not good, mainly because there are vehicles that are underloaded and cities that are unsatisfied. Moreover, it can be shown (see, e.g., Fig. 5) that the best-known solution of most of the problems proposed by [2] cannot be obtained by partitioning these problems into two sectors having approximately the same number of cities. To partition the problem in another way, it is necessary, after a summary resolution of the subproblems, to move some cities and some tours from one subproblem to another, while making use of the work performed up to that point.

After a summary resolution of the subproblem associated to each region (i.e., after having performed a given number of iterations of TS for each subproblem), one has a best solution that is constituted of formed tours, unsatisfied cities, and empty vehicles. These tours, cities, and vehicles have to be grouped in another way, forming therefore new subproblems. We use for this the following algorithm that is similar to the previous one but using the position of the center of gravity of tours instead of the position of the cities:

Subsequent partition of uniform problems:

(a) Compute the center of gravity of every tour (the cities have a weight of 1 and the depot a weight of 0).
(b) Sort the undelivered cities and the tours by angle of their center of gravity.
(c) Divide the problem into sectors, giving about the same number of tours to each sector. The tours are not divided and each undelivered city is attributed to the sector containing the tour for which the angle of the center of gravity is the closest to the angle of the city.
(d) Sort by radius of the center of gravity the undelivered cities and the tours of each sector.
(e) Divide the sectors into concentric regions in the same way as was done for the division into sectors.
(f) Assign at random the empty vehicles among the subproblems.

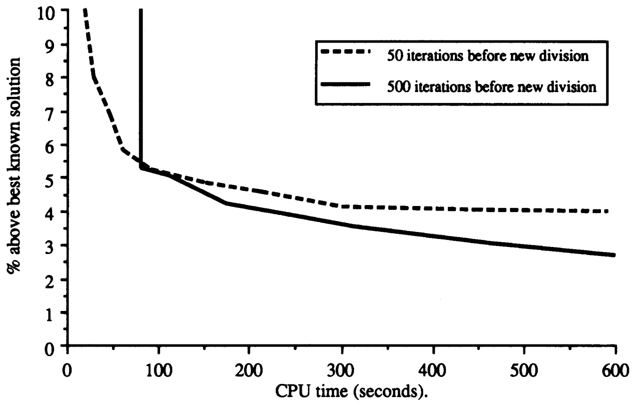

Fig. 3. Influence of the number of iterations performed before new division into subproblems.

The first sector needs not be constructed systematically by beginning the scanning with the tour or the city having the smallest angle of center of gravity. This tour or this city can be randomly chosen, for example. As assigning the empty vehicles to processes randomly seems convenient for the problems that we have considered, we have not tested other policies for point (f) of the algorithm.

In Figure 3, we present the behavior of this algorithm on a classical problem that involves 199 cities and was proposed in [2]. As the number of iterations performed at each summary resolution has to be chosen, we have plotted in Figure 3 the evolution of the value of the solutions as a function of the CPU time for two values of number of iterations: 50 and 500. The problem has been partitioned into four sectors and the parallel algorithm was implemented on the same sequential workstation as before; so the CPU time is the sum of the time required to solve every subproblem. As good solutions of this problem include 17 vehicles, it is not possible to partition this problem efficiently into more than four subproblems.

It turns out that the algorithm finds good solutions faster if 50 iterations are allowed for the resolution of subproblems, but it is difficult to find very good solutions during the further partitions of the problem. Conversely, if a large number of iterations is allowed for the resolution of the subproblems, it takes time to get a good solution, but the supplementary work performed during the further partitions of the problem permits finding much better solutions. It is why we propose to gradually increase the number of iterations performed by the search between two successive partitions and to perform $\lfloor 2d \cdot (n/r) \rfloor$ iterations at the dth decomposition of the problem (n is the total number of cities and r is the number of regions).

Figure 4 compares this algorithm to the automatic algorithm proposed by [6] and runs on the same com-

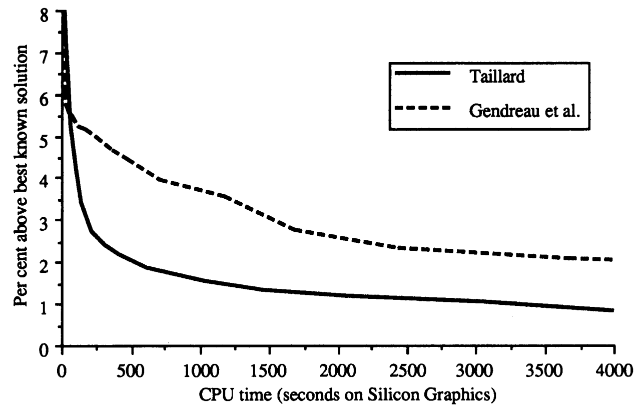

Fig. 4. Comparison of two TS implementations.

puter for the problem of [2] involving 199 cities. The automatic algorithm of [6] has been communicated to us by Hertz [10]. We have run our algorithm 12 times with different partitions of the problem into four parts. The other algorithm has been run with different initial random number generator seeds only 10 times, but this does not affect the comparison of both algorithms because we present *average* values in Figure 4. We see in this figure that our algorithm finds, after more than 1 min, better solutions on average than does the algorithm of [6]. It seems that the algorithm of [6] finds solutions that are relatively good in a short time. Indeed, for this problem and between 20 and 60 s of computation time, it finds on average better solutions than our algorithm. But, after 1 min, the algorithm of [6] seems to have difficulties to improve the quality of the solutions.

The initial partition has a great influence on the final solution produced by our algorithm; we have not found rules for choosing one initial partition rather than another. To illustrate the difficulty of finding a good initial partition, let us mention that, for this set of runs, the run that has found the best solution was the worst in terms of the solution produced after 2000 iterations and the run that has found the worst solution at the end was the best after 2000 iterations.

Table I gives the best value of solutions found by our algorithm, as well as the best published values for the problems proposed in [2]; the problems are listed in Table I in the same order as in this reference. Bold characters indicate best-known values. We see that we have improved five of 14 best solutions (and especially most of the largest problems were solved better) and that identical values of solutions were found for the remaining problems (optimum reached?). We do not give CPU times in Table I because it is very difficult to give them as our algorithm does not find the best-known solution of every problem at each run. We think that mentioning the times required for the best runs is not honest (in most cases, these times are smaller than the times needed to find solutions that are, on the average, 1% above the best-known solution [see Table II]). The fact that our algorithm has found every best-known solution should justify the foundations of the method.

In Table II, we give the CPU times (if available) required by our algorithm and by the algorithm of [6] to find solutions that are, on the average, 5, 2, and 1% above the best-known solution. To get these CPU times, we have run both algorithms for each problem instance 10 times or more with the same set of parameters but with a different random seed. Also, we have

TABLE I. Best-solution values produced by some TS implementations

No. Cities	Service Time/Longest Tour Limit L	Best Solution Value of [6]	Best Solution Value of [14]	Parallel TS (Partition into Sectors)
50	0/∞	**524.61**	**524.61**	**524.61**
75	0/∞	835.32	838.62	**835.26**
100	0/∞	**826.14**	829.18	**826.14**
150	0/∞	1031.07	1044.35	**1028.42**
199	0/∞	1311.35	1334.55	**1298.79**
50	10/200	**555.43**	**555.43**	**555.43**
75	10/160	**909.68**	**909.68**	**909.68**
100	10/230	**865.94**	866.75	**865.94**
150	10/200	**1162.55**	1164.12	**1162.55**
199	10/200	1404.75	1417.85	**1397.94**
120	0/∞	**1042.11**	**1042.11**	**1042.11**
100	0/∞	**819.56**	819.59	**819.56**
120	50/720	1545.93	1545.98	**1541.14**
100	90/1040	**866.37**	**866.37**	**866.37**

TABLE II. CPU time (in seconds on Silicon Graphics 4D/35) needed to find solutions that are, on average, a specified percent above the best-known value

		TS with Partition into Sectors			Implementation of [6]		
n	No. Sectors	5%	2%	1%	5%	2%	1%
50	1	7	**23**	**49**	6	53	>90
75	2	**3**	**12**	**53**	19	>110	—
100	2	12	68	580	**<5**	**9**	**53**
150	3	86	**450**	3800	**18**	>1800	—
199	4	**75**	**510**	**3000**	240	4600	—
50	1	**2**	**5**	**17**	12	21	>34
75	2	**5**	**20**	**51**	33	>99	—
100	2	**11**	**120**	**1100**	190	310	>1400
150	3	**64**	**400**	**1100**	550	>2800	—
199	4	**100**	**890**	—	1300	>2300	—
120	2	4600	—	—	—	—	—
100	2	81	170	340	**14**	**32**	**72**
120	2	**70**	**300**	3900	>2100	—	—
100	2	82	330	1500	79	280	**520**

taken the mean solution values obtained after a given time (e.g., the values given for the problem with 199 cities without longest tour length limitation can be read in Fig. 4). A time printed in bold characters indicates that the corresponding time for the other method is significantly larger.

We see in Table II that our algorithm finds solutions of given quality generally faster than does the algorithm of [6]. However, for two problems, the algorithm of [6] converges to very good solutions in a very short computation time. For most of the remaining problems, and especially for the largest ones and for those with longest tour length limitations, the algorithm of [6] tends to terminate before having reached a solution at 1 or 2% above the best-known solution. In this table, we see that the iterative search that we have designed for solving the subproblems is efficient: The problems with 50 cities have been solved without being partitioned; so our algorithm may be compared favorably to the algorithm of [6]. We see that the problem of [2] with 120 cities and without longest tour limitation is solved very poorly by both methods; for our method, this may be explained by the fact that the cities are grouped and by the fact that the depot is not centered: Indeed, an arbitrary decomposition of the problem into sectors has a high probability of splitting into different subproblems a group of cities that should be served by the same vehicle. Moreover, if the depot is not centered, an arbitrary decomposition has a high probability of creating one sector with a much larger angular span than that of the others. We will see in the next section how to partition such a class of irregular problems in a much more efficient way. Figure 5 gives the best solution found for the problems proposed by [2] including (a) 120 cities with longest tour limitation, (b) 150 cities without longest tour limitation, (c) 199 cities without longest tour limitation, and (d) 199 cities with longest tour limitation; the travel from the depot to the first city of each tour and the travel from the last city to the depot are not drawn on this figure.

To test our algorithm on bigger problems, we have generated problems for which we conjecture that the optimum solutions are known. These problems involve $(2q \cdot k)^2$ cities having a demand of 1 each and the capacity of each vehicle is $2q$. The rectangular coordinates of the cities are (i, j) $(i = 1 \ldots 2q \cdot k, j = 1 \ldots 2q \cdot k)$ and the coordinates of the depot are $(q \cdot k + 0.5, q \cdot k + 0.5)$. For $k = 1$, 2, and 3 and $q = 2$ and 3, what we conjecture to be optimum solutions are plotted in Figure 6; again, the travel from the depot to the first city of each tour and the travel from the last city to the depot are not drawn. Optimality has been conjectured because our TS algorithm has produced this type of solution for small values of k and we have not found any evidence for such solutions to not be optimum. For $q = 2$ and $q = 3$, we think that the optimum solutions are of this type for any positive integer k; but for $q \geq 4$, we know that better solutions with other structures exist.

Table III gives, for some values of k and q, the performances of our algorithm (in percent above the pseudooptimum value) for problems generated on a grid. In this table, s is the number of sectors and c the number of divisions of the sectors into concentric regions; therefore, $s \cdot c$ is the total number of subproblems. We give in this table the cumulative number of iterations performed by TS and the corresponding CPU times on a Silicon Graphics 4D/35 workstation.

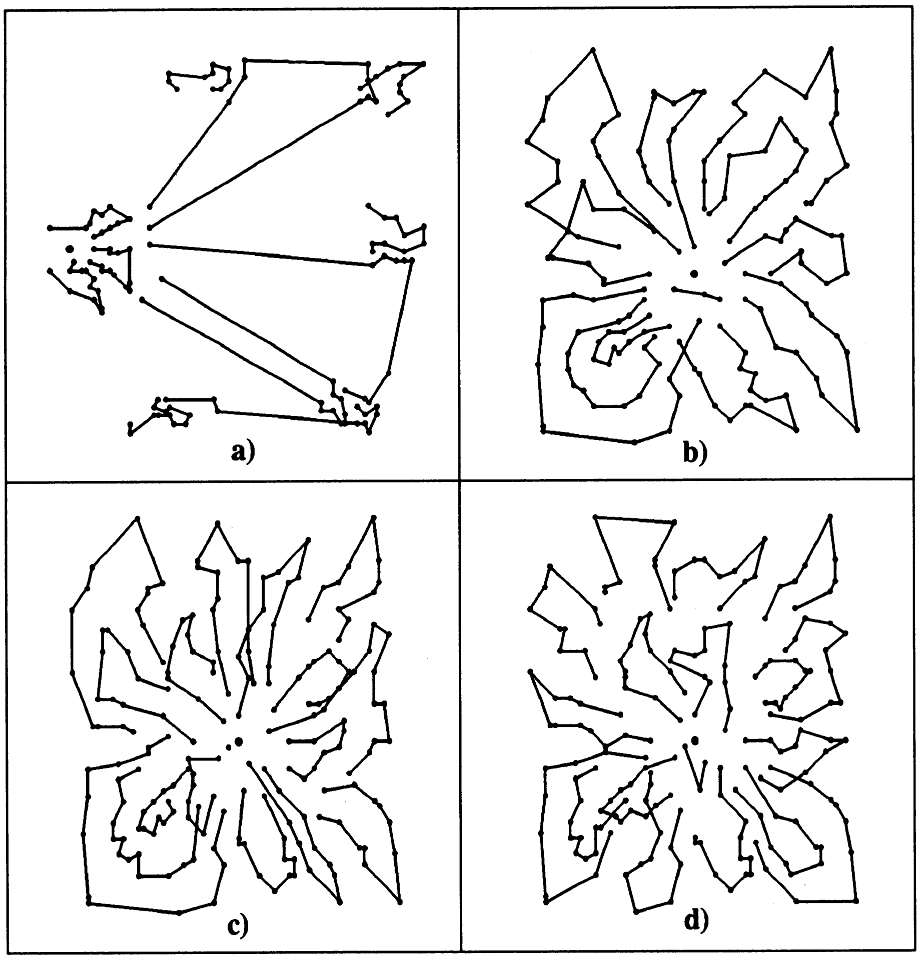

Fig. 5. Best-known solutions of some classical problems.

The automatic algorithm of [6] is much less efficient than ours for this type of problem: For the problem with 64 cities, after 150 s (respectively, after 700 s), the average solution value is 2.5% (respectively, 0.9%) above the pseudooptimum value; for the problem with 144 cities, after 770 s, the average solution value is

TABLE III. Parallel TS efficiency for partition into polar regions

No. Cities	No. Vehicles	*k*	*q*	*s*	*c*	No. Iterations	CPU Time (s)	% Above Pseudooptimum
64	16	2	2	2	1	150	1.7	2.4
64	16	2	2	2	1	2,000	17.9	0.9
64	16	2	2	4	1	150	0.7	2.8
64	16	2	2	4	1	2,000	6.2	1.7
256	64	4	2	4	1	500	7.9	1.6
256	64	4	2	4	1	3,000	94.8	0.9
256	64	4	2	2	2	500	8.5	1.8
256	64	4	2	2	2	3,000	96.7	1.2
1024	256	8	2	8	4	3,000	123.7	0.8
144	36	2	3	2	1	1,000	23.7	1.9
144	36	2	3	2	1	5,000	107.5	0.9
324	54	3	3	6	1	1,000	27.9	2.0
324	54	3	3	4	1	10,000	289.9	1.0

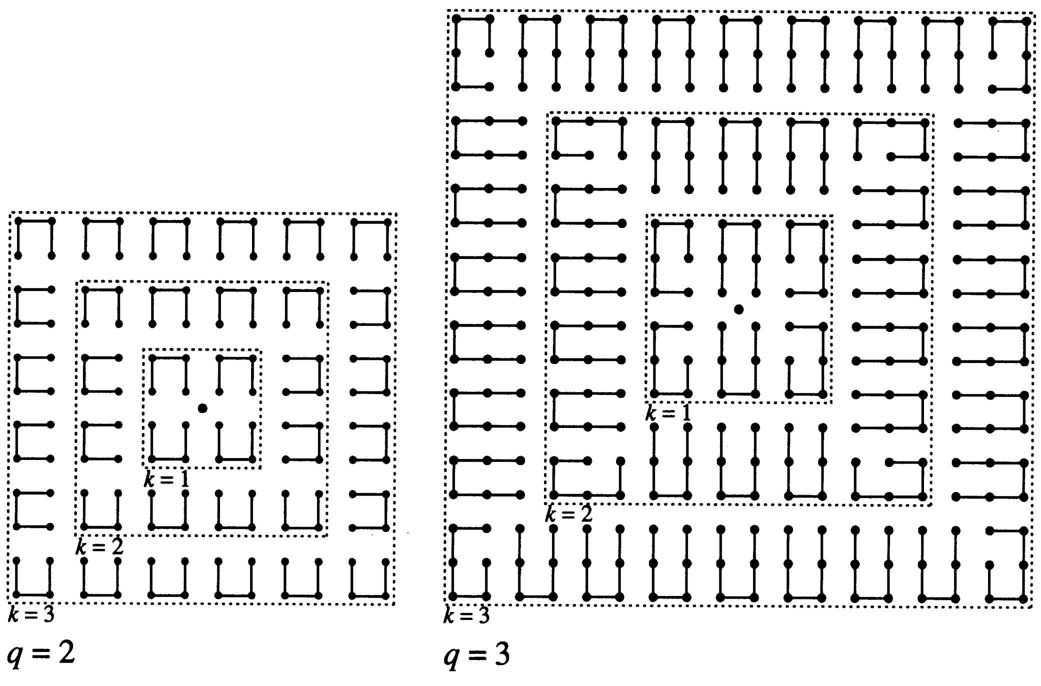

Fig. 6. Pseudo-optimum solutions on some grids.

more than 1.9% above the pseudooptimum value. The implementation of the automatic algorithm of [6] is not able to treat the other problems of Table III.

The implementation of our algorithm on distributed computers is easy. To each subproblem is assigned a process that has to communicate to the three or four processes assigned to adjacent regions (Fig. 7 shows the flows of information between processes). The division of the problem into regions occurs once. Then, every process transmits tours, undelivered cities, or empty vehicles to its neighbor process. For example, the number of tours that a process gives to its neighbor process placed at its right and receives from its left neighbor may be fixed between each summary resolution. The communications between processes located in the same sector must be done alternatively in direction of the depot and in the other direction, so that all the processes keep about the same sizes of subproblems.

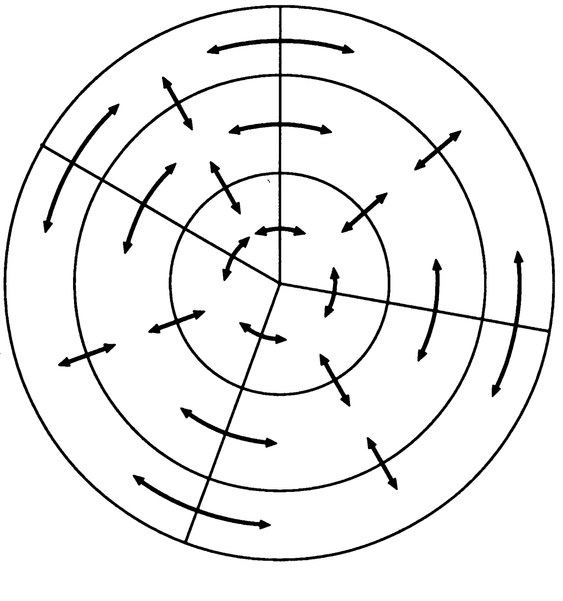

Fig. 7. Transfers of information between processes.

The most important limitation of the efficiency of such a parallelization is due to the difference of computing time between the slowest process (e.g., the one that has to deal with the largest subproblem) and the fastest. Experimentally, efficiencies of more than 80% may be reached if the problem is not partitioned into very small subproblems (let us say less than four vehicles per subproblem for the considered problems).

4. NONUNIFORM PROBLEMS

We have seen that our decomposition method does not work well if the cities are not regularly distributed around the depot or if the depot is not centered. Moreover, our method cannot be applied to non-Euclidean problems because it is based on the coordinates of the cities; in this case, we propose a decomposition method based on the partition of the arborescence of the shortest paths from the depot to all the cities. The intuition is that cities that are near to each other will probably belong to the same branch of the arborescence and thus will probably belong to the same subproblem.

Partition of non-Euclidean problems:

```
Data: $n$ cities, 1 depot.
    $d_{ij}$: distance between cities $i$ and $j$ ($i, j = 0 \ldots n$).
    $Q$: capacity of the vehicles.
    $q_i$: demand of each city ($i = 1 \ldots n$).
    $c$: maximal number of vehicles per subproblem.
Building the arborescence of the shortest paths:
  Construct the set of the shortest paths from the depot to each city; this set is an arborescence $\alpha$
  that has the depot as the root. Let $t_i$ be the total quantity of goods to deliver to the cities for which
  city $i$ is the root of a subarborescence of $\alpha$ (for a leaf, we have $t_i = q_i$; for the depot, we have
  $t_0 = \sum_{k=1}^{n} q_k$) and let $D_i$ be the depth of city $i$ in the arborescence (i.e., the number of arcs
  of the path from 0 to $i$).
Set $r = 0$ (number of subproblems created).
Decomposition of the arborescence:
  While $t_0 \neq 0$ repeat:
    Let $j$ be a city such that
      (1) $j$ is not assigned to a subproblem
      (2) $t_j \leq cQ$
      (3) $t_k > cQ$, where $k$ is the city (if any) connected to $j$ with $D_k = D_j - 1$
      (4) $D_j \geq D_{j'}$ for every city $j'$ meeting conditions (1) to (3)
      (5) $t_j \geq t_{j''}$ for every city $j''$ meeting conditions (1) to (4).
    Set $r = r + 1$
    Assign to subproblem $r$ the cities contained in the subarborescence having $j$ as root and that
    have not been assigned to another subproblem.
    Assign $\nu_r = \lfloor (t_j + 1.2)/Q \rfloor$ vehicles to subproblem $r$.
    Set $t_i = t_i - t_j$ for each city $i$ belonging to the shortest path from depot to $j$.
```

This decomposition of the problem may be viewed as the deletion of arcs in the arborescence α; a subproblem is then assigned for each connected component. In other words, this decomposition of the arborescence consists in deleting arcs in such a way that the subproblems successively created involve a quantity of goods as close to cQ as possible and as far from the root (in terms of number of arcs) as possible. The arcs deleted link the subproblems to each other in a natural way; it is easy to transfer nondelivered cities or tours from one subproblem to another.

This algorithm solves the following problem:

Given an arborescence having $n + 1$ vertices of weight q_i ($i = 0 \ldots n$, $q_0 = 0$) and a value cQ, suppress a minimum number of arcs of the arborescence such that each resulting connected component has a weight smaller or equal to cQ.

If the problem is Euclidean, the arborescence of the shortest paths is a star (i.e., every city is connected directly to the depot); in this case, the decomposition is not appropriate. Instead of taking the arborescence of the shortest path, we suggest taking the minimum weighted spanning arborescence, where the weight p_{ij} of an arc between i and j [of rectangular coordinates (x_i, y_i) and (x_j, y_j) and distant $\mathbf{d}_{ij} = \binom{x_j - x_i}{y_j - y_i}$ from each other] is given by the formula

$$p_{ij} = \begin{cases} \|\mathbf{d}_{ij}\| \left(1 + \mu \left(1 - \dfrac{\mathbf{d}_{ij} \cdot \mathbf{d}_{i0}}{\|\mathbf{d}_{ij}\| \cdot \|\mathbf{d}_{i0}\|}\right)\right) & \text{if } \|\mathbf{d}_{ij}\| \cdot \|\mathbf{d}_{i0}\| \neq 0 \\ \|\mathbf{d}_{ij}\| \text{ otherwise,} & \end{cases}$$

where $\nu \cdot \mathbf{w}$ denotes the scalar product of vectors ν and $\mathbf{w}$, and $\|\nu\|$ denotes the length of vector ν. The nonnegative parameter μ permits modification of the shape of the tree: $\mu = 0$ implies that $p_{ij} = \|\mathbf{d}_{ij}\|$; for $\mu > 0$, the more the angle between vectors $\|\mathbf{d}_{0i}\|$ and $\|\mathbf{d}_{ij}\|$ is away from π, the more the weight p_{ij} is greater than $\|\mathbf{d}_{ij}\|$.

Figure 8 shows graphically the influence of the parameter μ on the shape of the arborescence. The bigger μ is, the more the arcs of the minimum weighted spanning arborescence are in the direction of the depot. The set of points represents the towns and villages of the canton of Vaud in Switzerland; we chose the root at the École Polytechnique Fédérale de Lausanne (EPFL). Varying the parameters μ and c in this algorithm permits obtaining various partitions of the problem. Moreover, these partitions may be found very easily and efficiently; this is not the case for the decomposition method of [5] that requires solving an NP-hard problem (generalized assignment). The arborescence obtained with μ set to 1 or 2 looks (subjectively) very close to the arborescence of the shortest paths of the actual roads network. Although this partition algorithm has not been formally tested on non-Euclidean problems, we think that it should work well on real-life problems.

Thus, we divide irregular problems using this decomposition technique (choosing $\mu = 0.7$) and solving the subproblems as we did for the polar decomposition. However, we do not transmit cities or tours between resolutions any more (this is the reason for the experi-

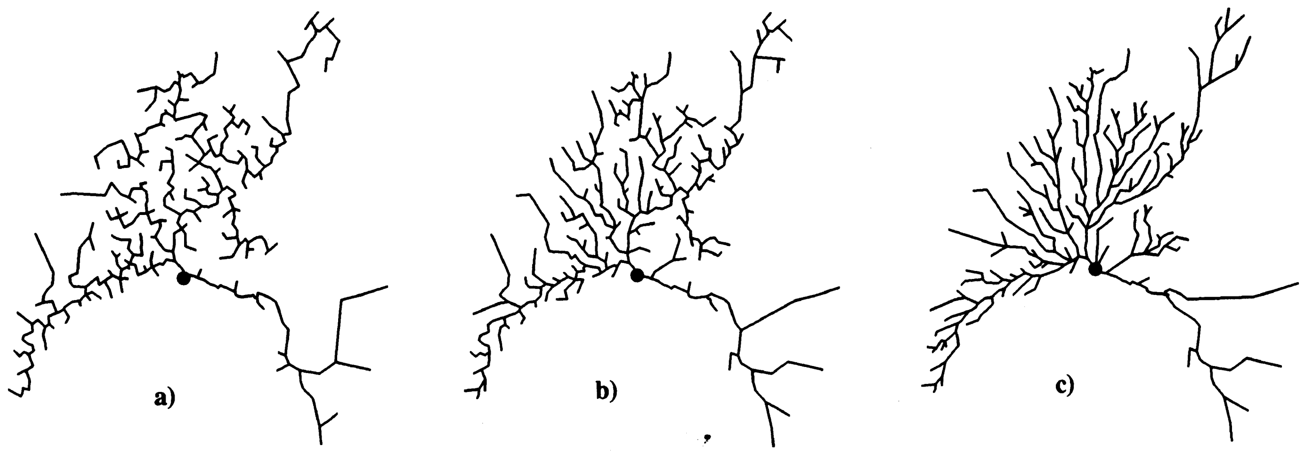

Fig. 8. Influence of parameter μ on the shape of the arborescence: (a) $\mu = 0$; (b) $\mu = 0.5$; (c) $\mu = 2$.

mental value 1.2 in the formula that fixes ν_r, the number of vehicles attributed to region r; this value is convenient for the instances of problems that we have considered; for other instances, it might be changed), but at the dth resolution, one starts with the best solution found at the $d - 1$th resolution and $(2dn)/r$ iterations of TS are performed.

Table IV compares our new algorithm to the previous one on irregular problems. We see that this new decomposition of the problems permits obtaining satisfactory solutions very rapidly. (The best-known solution value of the problem with 120 cities is 1042.11; the best-known solution value of the problem with 385 cities is 24599.6.) So, this decomposition method succeeds in finding a logical partition into subproblems.

The problem due to [2] is the one without longest tour limitation. The problem involving 385 cities has been generated as follows: Each city is the most important town or village of the smallest political entity (commune) of the canton of Vaud in Switzerland. The census of inhabitants of each commune has been taken at the beginning of year 1990. We have considered a demand of 1 unit per 100 inhabitants (but at least 1 for each commune) and vehicles of capacity 65. If a town has a demand of more than the capacity of the vehicles (hence, many vehicles have to visit this town), we have considered that all the vehicles except one will visit this town while being fully loaded. This means that the problem may be treated as if the demand q_i of city i that has a population of h_i inhabitants, is

$$q_i = \begin{cases} \left\lceil \frac{h_i}{100} \right\rceil \text{ if } \frac{h_i}{100} < 65 \\ max\left(1, \left\lceil \frac{h_i}{100} \right\rceil modulo\, 65\right) \text{ otherwise.} \end{cases}$$

$\lceil x \rceil$ denotes the biggest whole number smaller than $x + 1$. The depot was placed at the EPFL. The tours including only one city and performed by vehicles fully loaded have not been taken into account.

5. CONCLUSIONS

We have shown that it is possible to decompose VRP including a large number of vehicles into subproblems

TABLE IV. Partition into polar regions and partition of the arborescence

No. Cities	Origin	Solution Value (Polar Partition)	Solution Value (Arborescence)	CPU Time (s)
120	[2]	1463.2	1133.1	3.2
120	[2]	1266.1	1061.3	19.7
120	[2]	1243.1	1055.4	28.0
385	Pseudo-real	31216.0	26027.8	76.0
385	Pseudo-real	26534.3	25270.5	520.0

that may be solved independently. A decomposition into polar regions is appropriate for uniform problems, as demonstrated by the very high quality of the solutions found for all of the classical problems. However, we think that it is possible to more quickly find very good solutions by exploring other policies of data transmission between processes.

For nonuniform or non-Euclidean problems, another decomposition method is proposed and although the results obtained are acceptable, we think that policies of data transmissions between processes have to be studied in order to get very good solutions for such problems. For example, to imitate the partition into polar regions, one could build an arborescence with μ slightly smaller than 0 (-0.2) and the arcs joining two different subproblems and belonging to this arborescence might be interesting data transmission paths. Another possibility is to use the decomposition of the arborescence as a first decomposition of the problem and then to use the decomposition into polar regions for the subsequent decompositions of the problem. For non-Euclidean problems, interesting data transmission paths other than those belonging to the arborescence of the shortest paths might be problematic to determine. But, for real-life problems, the coordinates of the cities are known and it is possible to use, for the subsequent partitions of the problems, the decomposition into polar regions.

To test the efficiency of our method on large problems, we have proposed problems, generated on a grid, for which the optimum solutions seem known. Our methods, although not specially designed for such problems, rapidly find solutions at 1% above the pseudo-optimum solution values. Although we have not tested a partition into rectangular regions, as proposed by [11], we think that such a partition would be more appropriate for grid problems.

To conclude, we think that this paper outlines some principles that might be useful for creating new powerful software for the VRP.

The author would like to thank E. A. Silver whose valuable and multiple suggestions have improved the presentation of this paper. This research was supported by the Fonds national suisse pour la recherche scientifique, Grant Number 20-27926.89.

REFERENCES

[1] K. Altinkemer and B. Gavish, A parallel savings heuristic for the delivery problem with a log Q error guarantee. Graduate School of Management, University of Rochester, Rochester, NY (1985).

[2] N. Christofides, A. Mingozzi, and P. Toth, The vehicle routing problem. *Combinatorial Optimization* (N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, Eds.). Wiley, Chichester (1979) 315–338.

[3] G. Clarke and J. W. Wright, Scheduling of vehicles from a central depot to a number of delivery points. *Operations Res.* **12** (1964) 558–581.

[4] M. Desrochers and T. W. Verhoog, A Matching Based Saving Algorithm for the Vehicle Routing Problem. Cahier du GERAD G-89-04, École des Hautes Études Commerciales de Montréal (1989).

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

[6] M. Gendreau, A. Hertz, and G. Laporte, A tabu search heuristic for the vehicle routing problem. Report CRT-777, Centre de recherche sur les transports, Université de Montréal (1992) *Management Sci.* to appear.

[7] B. Gillett and L. Miller, A heuristic algorithm for the vehicle dispatch problem. *Operations Res.* **22** (1974) 340–349.

[8] F. Glover, É. Taillard, and D. de Werra, A user's guide to tabu search. *Ann. Operations Res.* **41** (1993) 3–28.

[9] F. Harche and P. Raghavan, A generalized exchange heuristic for the capacitated vehicle problem. Working paper. Stern School of Business, New York University (1991).

[10] A. Hertz, Private communication (Oct. 19, 1992).

[11] R. M. Karp, Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane. *Math. Operations Res.* **2** (1977) 209–244.

[12] A Marchetti Spaccamela, A. H. G. Rinnooy Kan, and L. Stougie, Hierarchical vehicle routing problems. *Networks* (1984) 571–586.

[13] R. H. Mole and S. R. Jamerson, A sequential route-building algorithm employing a generalised savings criterion. *Operational Res. Q.* **27** (1976) 503–511.

[14] I. H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Operations Res.* **41** (1993) 421–451.

[15] V. M. Pureza and P. M. França, Vehicle routing problems via tabu search metaheuristic. Publication CRT-747, Centre de recherche sur les transports, Montréal (1991).

[16] F. Semet and É. Taillard, Solving real-life vehicle routing problems efficiently using taboo search. *Ann. Operations Res.* **41** (1993) 469–488.

[17] D. Simchi-Levi and J. Bramel, On the optimal solution value of the capacitated vehicle routing problem with unsplit demands. Department of Industrial Engineering and Operations Research, Columbia University, New York, 1990 (revision 1991).

[18] É. Taillard, Parallel taboo search techniques for the job shop scheduling problem. Report ORWP 89/11 (revision 1992), Département de mathématiques, École Polytechnique Fédérale de Lausanne (1989). *ORSA J. on Comp.* To appear.

[19] P. Toth, Heuristic algorithms for the vehicle routing problem. Presented at the Workshop on Routing Problems, Hamburg (1984).

[20] A. Volgenant and R. Jonker, The symmetric traveling salesman problem and edge exchanges in minimal 1-tree. *Eur. J. Operational Res.* **12** (1983) 394–403.

[21] A. G. Willard, Vehicle routing using r-optimal tabu search. M.Sc. Dissertation, The Management School, Imperial College, London (1989).

Received April 1992
Accepted May 1993