Annals of Operations Research 41(1993)421–451

Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem

Ibrahim Hassan Osman
Institute of Mathematics and Statistics, The University, Canterbury, Kent CT2 7NF, UK

Abstract

The vehicle routing problem (VRP) under capacity and distance restrictions involves the design of a set of minimum cost delivery routes, originating and terminating at a central depot, which services a set of customers. Each customer must be supplied exactly once by one vehicle route. The total demand of any vehicle must not exceed the vehicle capacity. The total length of any route must not exceed a pre-specified bound. Approximate methods based on descent, hybrid simulated annealing/tabu search, and tabu search algorithms are developed and different search strategies are investigated. A special data structure for the tabu search algorithm is implemented which has reduced notably the computational time by more than 50%. An estimate for the tabu list size is statistically derived. Computational results are reported on a sample of seventeen bench-mark test problems from the literature and nine randomly generated problems. The new methods improve significantly both the number of vehicles used and the total distances travelled on all results reported in the literature.

Keywords: Local search, approximate algorithms, heuristics, hybrid algorithms, simulated annealing, tabu search, vehicle routing problem.

1. Introduction

The vehicle routing problem (VRP) under capacity and distance restrictions involves the design of minimum cost delivery routes for a fleet of vehicles, originating and terminating at a central depot, which serves a set of customers. Each customer is supplied by exactly one vehicle route. The total demand of any vehicle route must not exceed the vehicle capacity. The total length of any route includes the inter-customer travel times and service times must not exceed a prespecified bound. Figure 1 provides an illlustration of this type of problem.

The following notations are used for representing the problem:

n = the number of customers;
N = the set of customers, $N = \{1, \ldots, n\}$;
q_i = the demand of customer $i \in N$ ($i = 0$ denotes the depot, $q_0 = 0$);
δ_i = the service time of customer $i \in N$ ($\delta_0 = 0$);

© J.C. Baltzer AG, Science Publishers

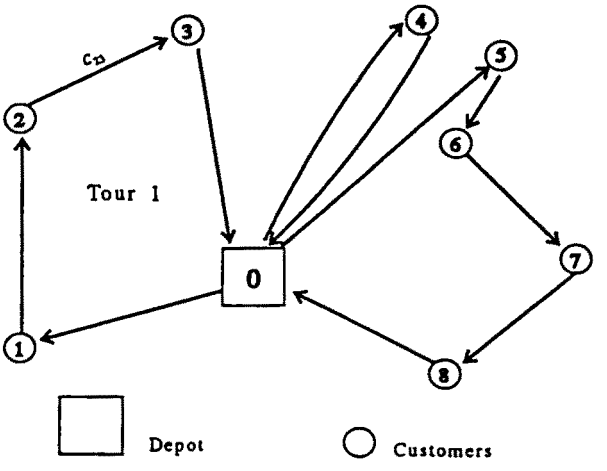


Fig. 1. The vehicle routing problem.

c_{ij} = the travel time (distance) between customers i and j, $c_{ij} = c_{ji}\ \forall i, j \in N$ ($c_{ii} = \infty, \forall i \in N$);

v = the number of vehicles, which is a *decision* variable in our problem;

V = the set of vehicles, $V = \{1, \ldots, v\}$;

Q = the vehicle capacity;

R_p = the set of customers serviced by vehicle p;

$C(R_p)$ = the cost (length) of the optimal travelling salesman tour π_p over the customers in $R_p \cup \{0\}$. This cost includes the travel times (c_{ij}) and the service times (δ_i);

L = the prespecified upper bound on the maximum tour length;

S = the feasible solution which is defined as $S = \{R_1, \ldots, R_v\}$;

$C(S)$ = the total sum of each individual tour length $C(R_p)$ for all $p \in V$.

Our goal is to find an optimal solution (say S without loss of generality) that minimizes the total travel length and satisfies:

$$\begin{aligned}
&\bigcup_{p=1}^{v} R_p = N, && R_p \cap R_q = \emptyset, \forall p \neq q \in V;\\
&C(R_p) = \sum_{i \in R_p \cup \{0\}} (c_{i\pi(i)} + \delta_i) \leq L, && \forall p \in V;\\
&\sum_{i \in R_p} d_i \leq Q, && \forall p \in V;\\
&C(S) = \sum_{p \in V} C(R_p), &&
\end{aligned} \tag{1}$$

where $\pi = \{\pi_1, \ldots, \pi_p, \ldots, \pi_v\}$ is an optimal TSP tour that minimizes the tour length for each $p \in V$.

The VRP is in an extremely active research area that has seen an exciting interplay between theory and practice. It is probably one of the greatest success stories of operations research. Numerous practical applications of the VRP are reported in the literature which reduced transportation costs for major companies from 6% to 15% (see, for instance, Brown and Graves [8], Fisher et al. [16], Bell et al. [5], Evans and Norback [15], Golden and Watts [26] for applications in the oil, chemical, food and drinks industries). Christofides [9], Bodin [7], and Golden and Assad [27] provide surveys of recent applications of the VRP.

Operational researchers' interest in the VRPs is partly due to their practical importance, but also to their intrinsic difficulties: as a generalisation of the travelling salesman problem (TSP), the VRP belongs to the class of *NP-hard* problems (Lenstra and Rinnooy Kan [33]), and polynomial time algorithms for finding optimal solutions are unlikely to exist. Hence, there have been few attempts to solve it optimally among such branch and bound procedures based on: a state space relaxation (Christofides et al. [12,13]), a TSP formulation (Laporte et al. [33]), and a set partitioning formulation (Agarwal et al. [2]). These approaches address small VRPs adequately up to 50 customers with 8 vehicles (Christofides [9]). Laporte and Nobert [32] provide a review of exact methods.

Due to the limited success of exact methods, considerable attention and research effort have been devoted to the development of efficient approximate algorithms (or heuristics) which can provide near optimal solutions for large-sized problems. These heuristics can be classified as follows: *Constructive heuristics* that gradually build up vehicle tours by inserting at each step a customer according to some savings measure until all customers are served. The savings algorithm of Clarke and Wright [14], which is the most widely used in practice, belongs to the class, many of its algorithmic improvements and variants have appeared in the literature (see, for instance, Gaskell [19], Mole and Jameson [36], Nelson et al. [37], Paessens [44], and Altinkemer and Gavish [3]). *Two-step methods* that are based on either cluster-first route-second or route-first cluster-second approaches. The cluster-first route-second methods identify clusters of customers assigned to vehicles and a minimum cost TSP tour for each cluster is computed (Gillett et al. [21], Christofides et al. [11], and Fisher et al. [17]). The route-first cluster-second methods build an optimal TSP tour and then partition it into feasible VRP routes (Beasley [4], Haimovich and Rinnooy Kan [28]). *Exact but incomplete tree search methods* that terminate before reaching optimality at feasible solutions (Christofides et al. [11]). *Improvement methods*, in which a given solution is iteratively improved by making local changes. Exchange procedures have been suggested for the TSP (Lin and Kernighan [35], Or [38], Johnson [29]) and for the VRP by Christofides and Eilon [10], Russell [45]. Stewart and Golden [46] use a Lagrangian relaxation to transform the VRP into a modified m-TSP and then applying an arc exchange procedure similar to Lin [34]. Bodin et al. [6], Golden and Assad [27], Osman [40] provide broad surveys and heuristic classification schemes.

This paper proposes simulated annealing (SA) and tabu search (TS) metastrategies, and investigates their algorithmic performances for the VRP under capacity and distance constraints. Computational results reveal that the proposed algorithms generate solutions that are significantly better than previously published solutions. Section 2 discusses iterative improvement methods based on *First-Improve* and *Best-Improve* selection criteria of neighbours which are generated by a new λ-interchange mechanism. Section 3 applies SA methodology using the cooling schedule proposed in Osman and Christofides [42]. Section 4 describes different TS implementations using special data structures and different selection strategies. In section 5, computational results are reported on seventeen bench-mark test problems from the literature and on nine randomly generated problems. Section 6 contains a summary and concluding remarks. Finally, the new best solutions obtained by our algorithms are provided in an appendix.

2. Iterative improvement methods

Most iterative improvement methods invoke the successive application of two modules: a construction method that produces an initial feasible solution S with a total tour length $C(S)$ as in the Clarke and Wright [14] procedure, and an improvement technique that maintains feasibility whilst reducing the tour cost iteratively. The latter consists of fundamental concepts: a generation mechanism to alter the initial solution; selection strategies of alternate solutions and a stopping criterion.

2.1. CLARKE AND WRIGHT SAVINGS (CW) PROCEDURE

The savings procedure of Clarke and Wright [14] is the most widely known heuristic for the VRP. The procedure begins with each customer being served by a single tour (fig. 2(a)). Cost savings $S_{ij} = c_{0i} + c_{0j} - c_{ij}$ can be obtained by satisfying the demands of customers i and j using one vehicle from the depot 0 (fig. 2(b)). These savings are sorted in decreasing order. The procedure merges customers i and j corresponding to the highest saving S_{ij} without violating the capacity restriction until no further merges are possible.

(a) Initial tours supplying i and j. (b) Combining i and j in a single tour.

Fig. 2. Cost savings.

2.2. λ-INTERCHANGE GENERATION MECHANISM

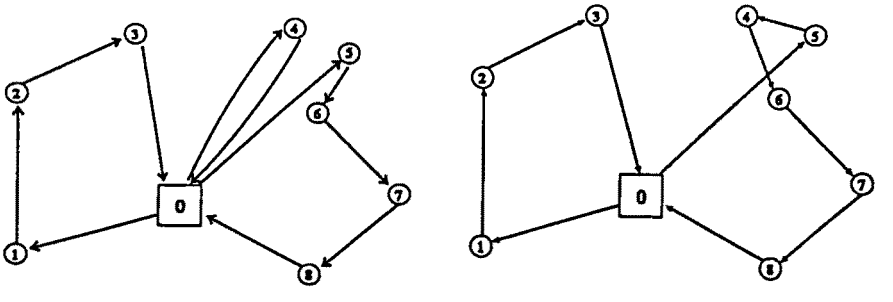
The generation mechanism describes how a solution S can be altered to generate another neighbouring solution S'. The λ-interchange mechanism has been defined in Osman [39], and used in Osman [41] and Osman and Christofides [42]. Here, we give an illustration on how this mechanism can be used for the VRP. Given a feasible solution for the VRP represented by $S = \{R_1, \ldots, R_p, \ldots, R_q, \ldots, R_v\}$, where R_p is the set of customers serviced by route p. A λ-interchange between a pair of route sets R_p and R_q is a replacement of a subset $S_1 \subseteq R_p$ of size $|S_1| \leq \lambda$ by another subset $S_2 \subseteq R_q$ of size $|S_2| \leq \lambda$, to get two new route sets $R'_p = (R_p - S_1) \cup S_2$, $R'_q = (R_q - S_2) \cup S_1$ and a new neighbouring solution $S' = \{R_1, \ldots, R'_p, \ldots, R'_q, \ldots, R_v\}$. The neighbourhood $\mathcal{N}_\lambda(S)$ of a given solution S is the set of all neighbours S' generated by the λ-interchange mechanism for a given λ (say, $\lambda = 1$ or 2).

The order in which neighbours are searched must be specified. Let the permutation σ be the order of vehicle indices in a given solution $S = \{R_1, \ldots, R_p, \ldots, R_q, \ldots, R_v\}$ (say, $\sigma(p) = p$, $\forall p \in V$), an *ordered search* selects all possible combinations of pairs (R_p, R_q) according to (2) and σ without repetition. A total number of $v(v-1)/2$ different pairs of routes (R_p, R_q) are examined to define a *cycle of search* in the following order:

$$(R_{\sigma(1)}, R_{\sigma(2)}), \ldots, (R_{\sigma(1)}, R_{\sigma(v)}), R_{\sigma(2)}, R_{\sigma(3)}), \ldots, (R_{\sigma(v-1)}, R_{\sigma(v)}). \quad (2)$$

Note that, for the descent and tabu search algorithm, the same permutation σ is used after each cycle of search is completed. Furthermore, for a given pair (R_p, R_q) we must also define the search order for the customers to be exchanged. We consider the case of $\lambda = 1$ and a similar analogy can be followed for other values of λ. The 1-interchange mechanism uses two processes to generate neighbouring solutions:

(i) A *shift process* which is represented by the (0, 1), (1, 0) operators. The (0, 1) and (1, 0) denote the shift of one customer from one route to another. Figure 3 shows an example of a (1, 0) shift process in which customer $4 \in R_p = \{4\}$ is removed and inserted into the route set $R_q = \{5, 6, 7, 8\}$. Note that the (1, 0) shift process

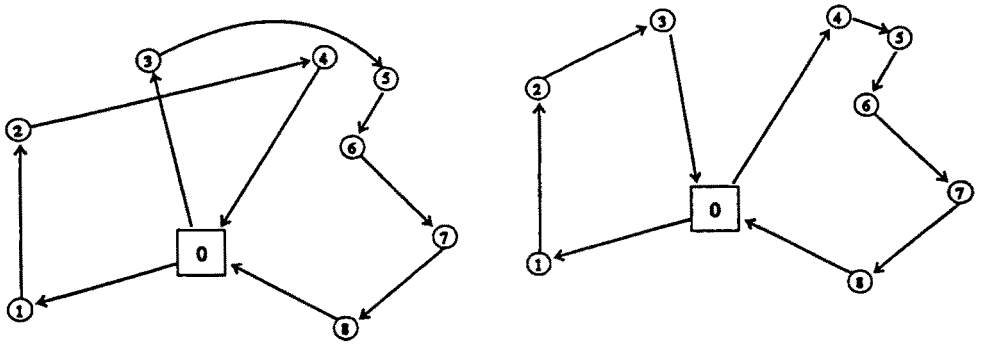


(a) Before the shift (b) After the shift

Fig. 3. A (1, 0) shift process.

would produce a new solution with an empty R_p and $R_q = \{5, 4, 6, 7, 8\}$. As a result, one vehicle route would be reduced. This is of great importance and is an important property of the generation mechanism.

(ii) An *interchange process* which is represented by the (1, 1) operator. This process exchanges each customer from one route with every other customer in another route. In fig. 4, we attempt to exchange systematically each customer in $R_p = \{1, 2, 4\}$ with every customer in $R_q = \{3, 5, 6, 7, 8\}$. Figure 4 shows an example where customer number 4 from route R_p is to be exchanged with customer number 3 from R_q (fig. 4(a)) to generate a new pair of routes (fig. 4(b)).



(a) Before the interchange (b) After the interchange

Fig. 4. A (1, 1) interchange process.

The customers in a given pair of routes are searched sequentially and systematically for improved feasible solutions by the shift and interchange processes. The order of search we implemented uses the following order of operators (0, 1), (1, 0) and (1, 1) on any given pair to generate neighbours.

2.3. EVALUATION OF THE COST OF A MOVE

A *move*, which is a transition from a solution S to a solution $S' \in \mathcal{N}_\lambda(S)$, may cause a change in the objective function values measured by $\Delta = C(S') - C(S)$. This change requires the evaluation of the length of the optimal TSP tours $C(R_p')$ and $C(R_q')$ generated by the λ-interchange mechanism from a given pair (R_p, R_q), with $p \neq q \in V$. A 1-interchange move may involve exchanging customer $i \in R_p$ with another $j \in R_q$, resulting in a change in the objective value of $\Delta = \Delta_{ij}$, where $\Delta_{ij} = C(R_p') - C(R_p) + C(R_q') - C(R_q)$. These $C(\cdot)$ values are computationally expensive to obtain. Therefore, two approximate methods are proposed for $C(\cdot)$ values and illustrated for $\lambda = 1$. This can similarly be generalised for other values of λ.

(a) *Insertion/deletion procedure*

Given π_p as the tour over $R_p \cup \{0\}$ and its tour length $C(R_p)$, let i be the customer to be inserted (or deleted) between two consecutive customers r and s in π_p and $l_i(r, s) = c_{ri} + c_{is} - c_{rs}$ be the cost of inserting i between r, s. If

$$l_i(R_p) = \min_{r,s \in \pi_p} \{l_i(r, s)\},$$

then the cost of the new generated tour over $R'_p \cup \{0\}$ becomes:

$$C(R'_p) = C(R_p) \pm l_i(R_p) \quad (+ \text{ if insertion, } - \text{ if deletion}).$$

The worst-case running time bound of this procedure is $O(n/v)$ since there are n/v customers on average in each tour.

(b) *2-opt procedure*

Perhaps the best known heuristic for the TSP is the arc exchange heuristic of Lin et al. [35]. The 2-opt procedure finds an initial random tour over the set of customers $R'_p \cup \{0\}$. This tour is improved by deleting two arcs, reversing one of the resulting two paths and reconnecting them until no additional improvement can be made. The worst-case running time of the 2-opt procedure is $O((n/v)^2)$.

(c) *Combination of procedures (a) and (b)*

Although the insertion/deletion procedure is fast, it may produce crossing of arcs in the new tour (see fig. 3(b)), in which case the 2-opt procedure is necessary to remove such crossing. A combination of the two procedures (a) and (b) provides a fast way to approximate the cost of exchanges. This combined procedure starts by evaluating each move by the insertion/deletion procedure; if a decision is made to accept a specific move, then the 2-opt procedure is invoked. Moves are evaluated thoroughly only if they seem worthwhile.

2.4. SELECTION STRATEGY OF ALTERNATE SOLUTIONS

In this paper, two selection strategies are used for choosing alternate solutions $S' \in \mathcal{N}_\lambda(S)$ when implementing iterative improvement methods:

(i) *Best-improve (BI) strategy*, which examines all solutions $S' \in \mathcal{N}_\lambda(S)$ in the neighbourhood of S and accepts the one which yields the best solution according to a given *acceptance criterion*.

(ii) *First-improve (FI) strategy*, which immediately accepts the first solution in the neighbourhood which satisfies the acceptance criterion.

2.5. THE λ-INTERCHANGE DESCENT ALGORITHM

The λ-interchange descent algorithm is an iterative improvement (or local search) method. It starts either with a solution S chosen at random or with the application of a constructive heuristic to reduce computing time and to generate a feasible solution. It then attempts to improve S by local perturbations using the λ-interchange mechanism to generate $S' \in \mathcal{N}_\lambda(S)$, which is selected according to FI or BI strategies and an acceptance criterion ($\Delta = C(S') - C(S)$, $\Delta < 0$). The search usually continues until a (local minimum) λ-optimal solution is found. A solution S is called *locally optimal* with respect to $\mathcal{N}_\lambda$ (*or* λ-opt for short) if and only if: $C(S) \leq C(S')\ \forall S' \in \mathcal{N}_\lambda(S)$. The algorithm steps are summarized below:

Step 1. Generate an initial heuristic solution S by the savings method.

Step 2. Choose a solution $S' \in \mathcal{N}_\lambda(S)$ in an ordered search and compute $\Delta = C(S') - C(S)$.

Step 3. If ($\Delta < 0$), then S' is accepted, set $S = S'$ and go to step 2.

Step 4. **If** a complete cycle of search – the neighbourhood $\mathcal{N}_\lambda(S)$ of S – has been searched without any improvements, **then** stop with a λ-opt solution, **else** go to step 2.

The above descent algorithm is denoted by 1 + FI if $\lambda = 1$ (2 + FI if $\lambda = 2$), which uses an ordered search of the neighbourhood and the FI selection strategy of neighbours. Similarly, 1 + BI represents a descent algorithm that uses the 1-interchange mechanism and the best-improve selection strategy in step 2. These algorithms are flexible and simple to implement. However, they have major limitations that the local optimum achieved may be from the global optimum and the quality of the final solution depends critically on the initial starting solution. In the next section, simulated annealing algorithms are used to overcome local optimality by embedding a randomized search and acceptance strategy into local search methods.

3. Simulated annealing implementation

The simulated annealing (SA) algorithm imposes different randomized search, acceptance and stopping criteria on the local search method in order to escape poor quality local minima. Local search descent methods do not accept non-improvement moves at any iteration, whereas SA does with certain probabilities. These probabilities are determined by a control parameter (T), called *temperature*, which tends to zero according to a deterministic *cooling schedule*. SA has its origin in statistical mechanics. The interest in SA began with the work of Kirkpatrick et al. [30], who proposed an SA algorithm based on the analogy between the annealing process of solids and the problem of solving combinatorial optimization. SA has been applied successfully to a large number of different combinatorial optimization problems, including the

flow-shop scheduling problem (Osman and Potts [43]); Osman and Christofides [42] for the capacitated clustering problem (CCP); Osman [41] for the generalised assignment problem (GAP). For more discussions on the theory and practical applications of SA, we refer to Aarts and Korst [1], and Osman [39].

We adopt for the VRP the non-monotonic SA cooling schedule introduced in Osman and Christofides [42], which requires specification of the following: (i) starting and final temperatures (T_s and T_f); (ii) decrement rule for updating the temperature T_k after each iteration k; (iii) update rule for temperature reset variables T_r after the system freezes; (iv) stopping criterion R, which is the total number of temperature resets to be performed after the best solution was found. This implementation uses the 1-interchange mechanism to generate neighbouring solutions. The neighbourhoods are searched sequentially in the order indicated in (2) according to different random permutations σ of the tour's indices $\{1, \ldots, v\}$. These permutations are generated each time a cycle search is completed. This is in constrast to the local search descent methods, where σ is fixed to an order of $\{1, \ldots, v\}$. Furthermore, the search for a given pair (R_p, R_q) is systematic for all potential customer moves as in the descent methods. This cooling schedule and its implementation is in contrast to classical SA schemes that have recourse to random neighbourhood search, which can lead to pockets that remain unexplored for undesirable lengths of time. The best solution found, S_b, during the search is kept rather than the one at which the SA algorithm stops. The algorithm performs a single iteration (one attempted feasible move) at each temperature. Our experience with similar implementations to the CCP and GAP shows that using the non-monotonic cooling schedule with an ordered search outperforms other SA in the literature with different cooling schedules and random selection of moves. Note that the importance of systematic neighbourhood search and a different type of non-monotonic search have been discussed by Glover [22] as basic features of TS methods. In this sense, our SA method consists of a hybrid of SA and TS ideas. Further details on these relationships can be found in Osman and Christofides [42], and Glover [25].

The hybrid SA/TS algorithm steps are as follows:

Step 1. Generate an initial heuristic solution S by the savings method.

Step 2. Initialisation of the *cooling schedule parameters*:
perform a *test* cycle of search over the neighbourhood $\mathcal{N}_1(S)$ of the initial solution without performing the exchanges in order to obtain the largest and smallest Δ_{max}, Δ_{min} change in objective function values, and an estimate of the total number of feasible exchanges *Nfeas*.
Set $T_s = \Delta_{max}$, $T_f = \Delta_{min}$, $T_r = T_s$, $\alpha = n \times Nfeas$, $\gamma = n$, $R = 3$, $S_b = S$ and $k = 1$.

Step 3. Select a solution $S' \in \mathcal{N}_1(S)$ in ordered search and compute $\Delta = C(S') - C(S)$ according to cost evaluation procedure (a).

Step 4. If $\{(\Delta \leq 0)$ or $\Delta > 0$ and $e^{(-\Delta/T_k)} \geq \theta$, where θ is a uniform random parameter $0 < \theta < 1\}$

then accept the new solution S', compute Δ according to cost procedure (b), set $S = S'$,

if $C(S') < C(S_b)$, then $S_b = S'$ and $T_b = T_k$, the temperature at which the best solution is found;

otherwise retain S.

Step 5. Update temperatures according to:

Normal decrement rule:

$$T_k = \frac{T_k}{(1 + \beta_k T_k)}, \quad \text{where} \quad \beta_k = \frac{T_s - T_f}{(\alpha + \gamma\sqrt{k})T_s T_f};$$

or

Occasional increment rule: If a cycle of search is completed without accepting any 1-interchange move, update as

$$T_r = \max\left\{\frac{T_r}{2}, T_b\right\} \quad \text{and set} \quad T_k = T_r.$$

Set $k = k + 1$.

Step 6. **Stop** if the stopping criterion is met (R resets were performed since S_b was found), report the best solution S_b and computation time.
Otherwise, go to step 3.

4. Tabu search implementation

Tabu search (TS) is a novel technique for solving combinatorial optimization problems. It is based on the general tenets of intelligent problem solving (Glover [23]). TS shares with SA the ability to guide iterative local search methods to continue the search beyond local optimality. The process in which the TS method seeks to transcend local optimality is based on an evaluation function which chooses the highest evaluation move in terms of objective function and tabu restrictions. This function selects a solution $S' \in \mathcal{N}_1(S)$ which produces the most improvement or the least non-improvement in the objective values at each iteration. By accepting non-improving moves, it becomes possible to return to solutions already visited, and tabu restrictions are to prevent such an occurrence. Further details on TS implementations and applications can be found in Osman [39,38] and Glover [23,24]. For any TS implementation, it is necessary to define the following:

(i) A *forbidding* strategy which manages what goes into the tabu list (list of tabu solutions).

(ii) A *freeing* strategy which manages what goes out of the tabu list.

(iii) A *short-term* strategy which manages the interplay between the above strategies including: an *aspiration* strategy which ignores tabu restrictions; a *selection*

strategy which chooses trial solutions from $\mathcal{N}_1(S)$ based on the best-admissible (BA) or the first-best-admissible (FBA) move selection strategies.

(iv) Stopping criterion.

In addition, longer term strategies are relevant to a variety of applications (see, for example, refs. [23,24,41]).

4.1. THE FORBIDDING STRATEGY

This strategy constrains the search by classifying certain moves as *forbidden* (or tabu) based on *tabu conditions* which are identified by the *attributes* of a move. To avoid cycling, it is sufficient to check that previously visited solutions are not revisited, but this requires a great deal of memory and computational effort. A data structure for the tabu list will be used to store a partial range of solution attributes rather than the complete visited solutions.

The tabu list data structure, *TABL*, takes the form of an $(n+1) \times v$ matrix (n rows, one per customer, one for the *null* customer involved in the shift process (0, 1) or (1, 0), and v columns, one for each route set R_p)). A move may consist of two pairs (i, R_p) and (j, R_q) which identify that a customer i from the set R_p of customers on route p has interchanged with a customer j from the set R_q of customers on route q, and vice versa. The attributes (i, R_p) and (j, R_q) specify *tabu restrictions* that forbid a move being performed. A move is deemed tabu if i is returned to R_p *and* j is returned to R_q. This is an approximation to forbid moves and the advantage is that more solutions can be represented and checked faster. $TABL(i, p)$ records the iteration number at which a customer i is removed from the route set R_p. Initially, the matrix *TABL* is initialised with high negative values to avoid false identification of customers as tabu during the initial iterations.

4.2. THE FREEING STRATEGY

This strategy is concerned with the management of what goes out of the tabu list after $|Ts|$ iterations, where $|Ts|$ is known as the tabu list size. The $|Ts|$ value is determined, as explained later, by a function depending on problem characteristics and selection of strategy of moves. The set of forbidden moves is recorded in the tabu list for a period of $|Ts|$ iterations. A simple and fast tabu status check is of great importance, especially when problem and tabu list sizes increase. At iteration k, a move is classified as tabu if neither i should return to R_p nor j should return to R_q during the following $|Ts|$ iterations. That is,

$$k - TABL(i, p) < |Ts| \quad \text{and} \quad k - TABL(j, q) < |Ts|. \tag{3}$$

Since *TABL* stores the iteration numbers, the tabu status of a potential move can be checked using the two simple operations in (3). With *TABL*, the tabu status of previous moves are updated automatically, as opposed to the classical circular tabu list approach which needs more input control from the freeing strategy.

4.3. THE SHORT-TERM STRATEGY

The short-term strategy forms the core of the TS algorithm. It is designed to permit the evaluation of the best admissible move in the neighbourhood based on tabu restrictions and aspiration criteria. A move is considered admissible if it is a non-tabu move, or a tabu move which passed an aspiration level criterion. Tabu restrictions and aspiration criteria play a dual role in constraining and guiding the search process (Glover [23]). In the tabu list, we store some attributes of moves to represent solutions. Thus, some non-tabu solutions may be prevented by tabu restriction due to this approximation and aspiration criteria are tests to correct such prevention. The following aspiration function will be used, which allows a new direction of search and guarantees no cycling. Let S_b be the current best solution found so far during the search. Let $S' \in \mathcal{N}_1(S)$ be a tabu solution. The new solution S' is admissible if $C(S') < C(S_b)$.

Two selection strategies will select an admissible move from the *candidate list* of moves: the *best-admissible* selection strategy, BA, and the *first-best-admissible* strategy, FBA. The BA strategy selects the best admissible move from the current neighbourhood which yields the greatest improvement or the least non-improvement in the objective function. The TS algorithm that uses the BA selection strategy is denoted by TS + BA. The FBA strategy combines a greedy approach with the BA strategy. It selects the first admissible move that provides an improvement in the objective value over the current solution; if all moves in the candidate list are tried without any improvement, then FBA selects the best recorded non-improving move. The TS algorithm that uses the FBA selection strategy is denoted by TS + FBA. The candidate list for the TS + FBA algorithm is the whole neighbourhood $\mathcal{N}_1(S)$ and its size is dynamic and determined automatically by the search itself. This dynamic sampling is a desirable way to search a large neighbourhood. However, the candidate list of moves for the BA strategy is the whole neighbourhood $\mathcal{N}_1(S)$ and its size is fixed. This list is very expensive to compute for large-sized problems because $\mathcal{N}_1(S)$ must be re-evaluated to select the best move after each iteration. Thus, we propose a data structure which allows only a small number of re-evaluations in order to identify a *new best* move from one iteration to another.

4.3.1. The special data structure (DS) for the BA selection strategy

The candidate list data structure (DS) can be briefly described as follows: BSTM and RECM are two matrices with dimensions $v \times v$, $\{v(v-1)/2\} \times 2$. The top triangular part BSTM(p, q) $(1 \leq p < q \leq v)$ is used to store the change in the

objective value Δ_{ij} associated with the best move obtained, exchanging customer $i \in R_p$ with $j \in R_q$, or an arbitrary high value if such a best move is not allowed. The lower triangular part BSTM(q, p) is used to store a positional index l associated with the pair R_p and R_q in the set of possible pair combinations $\{1, \ldots, v(v-1)/2\}$. An index indicates the position where the attributes of the best move are stored, for instance, RECM$(l, 1) = i$, RECM$(l, 2) = j$.

DS evaluates all moves in the neighbourhood $\mathcal{N}_1(S)$ only once at the first iteration. During the search, the upper matrix of BSTM is scanned for the best Δ_{ij} and the corresponding index l of the route sets is identified and used to obtain the attributes of the best move from the data matrix RECM. Such an accepted move involves R_p and R_q sets, only the other route sets remain intact. As a result, only moves in $2 \times v$ pair combinations of route sets (R_p, R_m), $\forall p \neq m$, and (R_q, R_m), $\forall q \neq m$, need to be evaluated rather than all moves in $v(v-1)/2$ pair combinations without DS. Figure 5 shows the increase in the number of combinations examined for problems with $4 \leq v \leq 20$ with DS and without DS. The advantage of DS is that it saves computation time without sacrificing the quality of the solution.

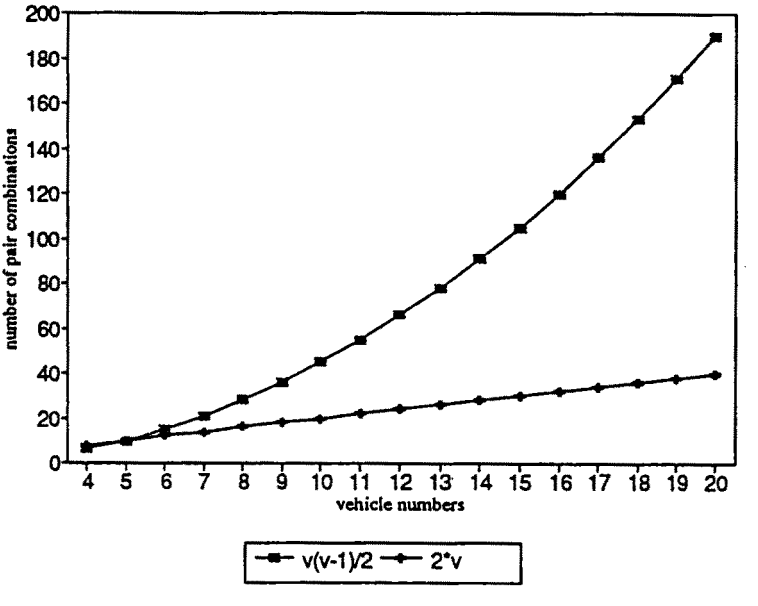


Fig. 5. Computational requirements of $2 \times v$ with DS and $v(v-1)/2$ without DS for each iteration of the BA strategy.

Note that the 1 + BI descent algorithm can make use of the DS data structure and the resulting algorithm is represented by 1 + BI + DS. However, the TS + FBA algorithm can not make use of this data structure since the size of the neighbourhood between iterations is variable and determined by improving moves. The candidate

list size is fixed in the case of the TS + BA algorithm, whereas it is variant in the TS + FBA algorithm. The TS + FBA algorithm records and updates the best non-improving admissible move during the search for the following reason. If we search the whole neighbourhood without finding any improved solution over the current one, then the best admissible move is accepted. At this moment, the TS + FBA is similar to the TS + BA algorithm and has the same neighbourhood size. Moreover, the TS + FBA algorithm accepts possibly more moves in good regions, updating the tabu list more frequently and searching over a larger part of the solution space.

4.3.2. The tabu list size functions

The tabu list size $|Ts|$ depends on problem characteristics (customer number n, vehicle number v, problem tightness ρ, which is the capacity ratio of the required demands to the available vehicle capacities) and the selection strategies (FBA and BA). A good estimate ts of $|Ts|$ was obtained using the experimental data in table 4 as follows:

Regressing the "Tabu size" values in column 4 of table 4 on problem sizes n, vehicle numbers v, and capacity ratio ρ in table 1, for the case of the TS + FBA algorithm, the ts value can be estimated by:

$$ts = 8 + (0.078 - 0.067 \times \rho) \times n \times v. \tag{4}$$

Similarly for the case of the TS + BA algorithm, regressing the "Tabu size" values in column 9 of table 4 on n and v to obtain an estimate of ts is given by:

$$ts = \max\{7, -40 + 9.6 \times \ln(n \times v)\}. \tag{5}$$

Since the $|Ts|$ value is statistically estimated, an error might occur. The $|Ts|$ value is then varied to take in a systematic order each of the three values $0.9 \times ts$, ts, and $1.1 \times ts$ and retains it for $2 \times |Ts|$ iterations before it is assigned another value. If all three values are chosen, a random order of the three values is obtained and the assignment is restarted. In similar experiments, Taillard [47] shows that varying $|Ts|$ randomly to take a value inside a given interval has an advantage.

4.4. THE STOPPING CRITERIA

The stopping criteria used in TS algorithms is based on a maximum number of iterations (*MAXI*) after the best solution has been found. This has the obvious advantage of relating the stopping criterion to solution changes at the cost of greater computational effort.

Multiple regression analysis was used to identify the minimum desired number of iterations M needed to obtain good solutions using the best iteration numbers at which the best solutions were found. A fitted equation to obtain an approximate

value for M is similarly estimated like the $|Ts|$ value. This is merely a guidance so that extra time can be saved and good solutions can be obtained with a reasonable computation time. A good fit was obtained with $R^2 = 81.8$ regressing the "best iteration" numbers (column 10 of table 4) at which the best solutions were obtained on problem characteristics for the case of the TS + BA algorithm. M was then estimated by:

$$M = 340 + 0.000353 \times \rho \times (n \times v)^2. \tag{6}$$

The TS general algorithm steps are as follows:

Step 1. Given an *initial heuristic* solution S by the savings method, perform a cycle of search to initialise the DS matrices BSTM and RECM if the BA strategy is used.
Set a value for *tabu list size* $|Ts|$, a high value for *TABL the initial tabu list*, a value for *MAXI* (or a value for M) and $S_b = S$ the best solution so far. Set $k = 1$ and $k_b = 0$.

Step 2. Choose a feasible and admissible move $S' \in \mathcal{N}_1(S)$ according to the **BA** or **FBA** selection strategies. Store the attributes of the newly accepted move in *TABL*. Update the current solution $S = S'$ and set $k = k + 1$.
If $C(S') < C(S_b)$, update the best solution $S_b = S'$ and set $k_b = k$.
If using the BA strategy, update (DS) the data structure BSTM and RECM matrices.

Step 3. **If** $(k - k_b > MAXI)$, go to step 4. **Otherwise**, go to step 2.

Step 4. Stop, report the best solution S_b with computation time.

5. Computational experience

Our aim in this section is to assess the effectiveness of the developed algorithms. The algorithms were tested on seventeen standard problems from the literature. Problem sizes range from 29 to 199 customers with tight and loose capacities, with and without maximum length constraints. Nine randomly generated new problems are added to the list with sizes equal to 50, 75 and 100 customers. Coordinates are taken from a $U[1, 100]$, while depot coordinates are chosen from a $U[45, 55]$. Customer demands are generated in the interval $U[20, 40]$, while vehicle capacity is fixed so that ρ, the ratio of the total required demands to the total available capacities, is in $U[0.90, 0.92]$. This information is summarised in table 1.

In the literature, data with customer locations defined by coordinates were published and the calculation of Euclidean distances is assumed between the customers. This could be done in a real-valued floating point operation or in an integer-valued operation, whereby the decimal fraction is rounded or truncated. Different solution values were reported without the sequence of routes; therefore, statements about the

Table 1

Characteristics of test problems.

Problem number[a]	Problem size	Vehicle capacity	Maximum tour length	Service time	Capacity ratio	Best published[b]	$C(S_{best})$ new best solution[c,d]
G1 [10]	29	4500	240	10	0.70	975/4 [10]	875/4
CL [14]	30	140	∞	0	0.92	1214/7 [45]	**1205/7**
G2 [19]	32	8000	240	10	0.91	810/4 [10]	810/4
C1 [10]	50	160	∞	0	0.97	524/5 [17]	524/5
C2 [10]	75	140	∞	0	0.97	855/10 [3]	**838/10**
C3 [10]	100	200	∞	0	0.91	833/8 [17]	**829/8**
C4 [11]	150	200	∞	0	0.93	1082/12 [11]	**1044/12**
C5 [11]	199	100	∞	0	0.98	1351/17 [3]	**1334/16**
C6 [10]	50	160	200	10	0.80	560/ [17]	**555/6**
C7 [10]	75	140	160	10	0.88	916/12 [17]	**909/11**
C8 [10]	100	200	230	10	0.81	885/9 [17]	**866/9**
C9 [11]	150	200	200	10	0.80	1210/15 [3]	**1164/14**
C10 [10]	199	200	200	10	0.88	1464/19 [3]	**1417/18**
C11 [11]	120	200	∞	0	0.98	1046/7 [44]	**1042/7**
C12 [11]	100	200	∞	0	0.90	822/10 [44]	**819/10**
C13 [11]	120	200	720	50	0.62	1551/11 [3]	**1545/11**
C14 [11]	100	200	1040	90	0.82	874/11 [3]	**866/11**
N1 New[e]	50	275	∞	0	0.90	–	709/6
N2 New	50	195	∞	0	0.90	–	814/8
N3 New	50	165	∞	0	0.90	–	994/10
N4 New	75	350	∞	0	0.91	–	925/7
N5 New	75	265	∞	0	0.90	–	1045/9
N6 New	75	223	∞	0	0.91	–	1011/11
N7 New	100	410	∞	0	0.90	–	1035/8
N8 New	100	330	∞	0	0.91	–	1185/10
N9 New	100	266	∞	0	0.91	–	1234/12

[a] Numbers in brackets represent the reference of problem origin.

[b] *a/b* [·]: *a*: solution value; *b*: number of vehicles; [·]: reference in which the solution was obtained.

[c] *a/b*: *a*: real-valued solution, *b*: number of vehicles, that were obtained in this study.

[d] Boldface indicates a better solution value or a smaller number of vehicles were found.

[e] New randomly generated problem.

kind of calculation of the Euclidean distances can not be made. Also, the published vehicle numbers are assumed to be equal to the previously known in cases where neither the vehicle number nor the sequence for every route are given. Our distances are calculated in a real-valued operation and the full solutions of our results are provided in an appendix to help other researchers. Our real-valued solutions are also provided, which are significantly better in terms of solution quality and number of vehicles than all the best published real solutions in the literature.

The algorithms are programmed in FORTRAN 77 and run on a VAX 8600 computer. The average computation time (ACT) in CPU seconds of the actual execution is reported excluding input and output time. The average relative percentage deviations (ARPD) of the objective function value $C(S)$ over the *new best solutions* S_{best} in table 1, i.e. $ARPD = (C(S) - C(S_{best}))/C(S_{best}) \times 100$ are also reported.

5.1. DESCENT ALGORITHMS

Sensitivity analysis using various neighbourhood sizes, selection strategies of alternate solutions and cost evaluation procedures was performed to examine the impact on running time and solution quality. In effect, we tested the effect of neighbourhood size as produced by 1-interchange and 2-interchange mechanisms using the FI selection strategy in the 1 + FI, 2 + FI descent algorithms with the 2-opt cost evaluation procedure (b). The BI selection strategy is only implemented using 1-interchange in the 1 + BI descent procedure with the same 2-opt cost procedure (b). Furthermore, the BI strategy is implemented using the proposed data structure (DS) with the combined cost evaluation procedure (c) in the 1 + BI + DS descent algorithm, which also used the 1-interchange neighbourhood mechanism.

Table 2 provides computational results in terms of solutions obtained and in CPU seconds. In evaluating the results, we observe that the Clarke and Wright [14] (C&W) algorithm produces poor initial solutions with an ARPD of 26.65%, varying from 0.87% to 28% for the published data and 28% to 59% for the random data using a total of 170 vehicles. The 1 + FI and the 2 + FI algorithms improve significantly the initial starting solutions of C&W and the ARPD is reduced to 10.07% and 4.87% at an increase in ACT from 1.51 to 141.22 and 2941.22 CPU seconds, respectively. The 1 + FI algorithm seems to perform better than the 2 + FI algorithm with respect to computation time. However, the latter identified the best solutions for small-sized problems G1 and G2. The 2 + FI also shows a great variability in its computational requirements; it takes 14,886 seconds for C11 of size 120 but only 4,274 seconds for C5 of size 199, although they have the same capacity ratio of 0.98%. The reason for this computational variability is partially due to the large neighbourhood search needed if at least one improvement has occurred over the 1 + FI solutions. Furthermore, the 1 + FI algorithm has started from poor C&W solutions with ARPDs of 24% and 28% for problem C11 and C5, respectively, thus requiring a large number of iterations to find good solutions. Descent algorithms are heavily dependent on good initial starting solutions and a good neighbourhood search mechanism to save computation time and obtain acceptable solutions.

In terms of selection strategy, the 1 + FI algorithm performs well: table 2 shows an ARPD of 10.07%, quite close to the 9.91% for the 1 + BI algorithm (without the data structure), although the latter requires an ACT of 582.42 seconds (four times more than the ACT of the 1 + FI algorithm). Finally, table 2 reveals that the 1 + BI + DS algorithm (with data structure and cost procedure (c)) improves the ACT of the 1 + BI algorithm by 2390% and that of the 1 + FI by 504% with an

Table 2

Computational results for the λ-interchange descent methods.

Problem number	C&W[a] value	CPU time	1 + FI[c] value	CPU time	2 + FI[c] value	CPU time	1 + BI[c] value	CPU time	1 + BI + DS[c] value	CPU time	Best published
G1	1017/5	0.1	953	2.2	875[b]	60.6	953	5.6	953	0.5	875
CL	1258/7	0.1	1255	0.2	1255	0.8	1255	0.2	1255	0.1	1214
G2	888/5	0.1	833	3.1	810[b]	36.1	833	5.2	833	0.4	810
C1	625/5	0.2	588	12.9	588	52.1	588	18.2	592	1.1	524
C2	1005/10	0.7	864	5.0	859	71.9	871	17.5	871	2.4	855
C3	982/8	1.1	942	46.3	910	15.0	962	91.2	956	5.6	833
C4	1299/12	3.3	1186	137.2	1127	4028.5	1158	410.0	1165	22.7	1082
C5	1707/17	7.0	1515	205.4	1421	4274.9	1474	705.4	1519	20.2	1351
C6	670/6	0.3	604	15.2	568	474.6	570	65.9	584	4.8	560
C7	989/12	0.7	964	24.1	953	190.3	953	51.4	947	9.4	916
C8	1055/10	1.2	980	146.0	923	7620.0	966	795.3	978	27.2	885
C9	1383/15	3.1	1298	423.0	1282	7094.0	1293	2475.0	1294	149.1	1210
C10	1671/20	7.4	1550	786.0	1504	13100.0	1535	54720.0	1542	240.5	1464
C11	1291/7	2.2	1104	555.4	1053	14886.3	1086	1824.9	1167	10.2	1046
C12	939/10	1.0	834	159.1	834	736.0	877	150.0	887	3.4	822
C13	1646/11	2.2	1620	292.0	1564	9205.0	1613	1062.0	1620	43.2	1551
C14	952/11	1.1	884	91.0	884	558.6	885	203.2	885	12.3	874
N1	948/6	0.2	754	13.6	729	148.5	765	21.7	811	1.4	709
N2	1134/8	0.2	897	9.6	876	38.4	966	10.9	932	2.6	814
N3	1281/10	0.2	1061	3.7	1010	18.0	1100	8.9	1122	2.3	994
N4	1476/7	0.7	1072	87.3	991	1075	74.0	1287	3.9	925	
N5	1534/9	0.6	1277	24.8	1143	538.5	1287	50.2	1352	4.5	1045
N6	1532/11	0.7	1169	18.6	1052	207.1	1126	69.9	1306	5.5	1011
N7	1579/8	1.5	1147	483.3	1133	4277.1	1174	1056.0	1341	14.4	1035
N8	1704/10	1.6	1412	88.0	1196	5010.8	1360	290.0	1512	12.0	1185
N9	1902/12	1.6	1480	38.0	1295	842.3	1435	203.7	1658	7.0	1234
ARPD[d]	26.65	–	10.07	–	4.87	–	9.91	–	14.36	–	–
ACT	–	1.5	–	141.2	–	2941.4	–	582.2	–	23.3	–

[a] C&W is the Clarke and Wright [14] algorithm, a/b: a: solution value $C(S)$, b: number of vehicles v.

[b] Best published solution value was obtained.

[c] 1 + FI: Descent algorithm with 1-interchange mechanism, First-Improve selection strategy and move calculation procedure (b).
2 + FI: Descent algorithm with 2-interchange mechanism, First-Improve selection strategy and move calculation procedure (b).
1 + BI: Descent algorithm with 1-interchange mechanism, Best-Improve selection strategy and move calculation procedure (b).
1 + BI + DS: Descent algorithm with 1-interchange mechanism, Best-Improve selection strategy and move cost procedure (c) and proposed data structure (DS).

[d] ARPD: Average relative percentage deviations over the new best obtained solutions for all test problems.
ACT: Average computation time in CPU seconds.

ARPD of 14.36%, which is worse than that of the 1 + BI algorithm by 44%. Consequently, the 1 + FI algorithm gives better results than the 1 + BI + DS algorithm, but the latter requires less computation time.

5.2. METASTRATEGY ALGORITHMS

In this section, we evaluate the performance of SA and TS algorithms using the same test problems. The SA algorithm and its cooling schedule are superimposed on the 1 + FI descent algorithm with the cost evaluation procedure (c). Computational results of SA are listed in table 3. Results show that the SA algorithm finds 10 new better solutions and two equal solutions to the previously best published solutions identified with [b] and an asterisk * in table 3, respectively. SA fails to reach the previously best known solutions for the rest of the problems with tight capacities. The SA solution quality is not robust and varies with problems, making solutions found by the 2 + FI descent algorithm better in some cases; for instance, the 2 + FI solution for C14 is 10.5% away from the best known solution compared to 13% for that of the SA algorithm. In addition, the SA solutions were worse than the 2 + FI solutions for six out of nine random problems, although the algorithm that performed the best also uses the longest computation time. However, SA solutions can be improved by further tuning of its parameters for problems where running time is short. Table 3 also provides the CPU time "time to best" to the "best iteration" numbers at which the best solutions were found, together with the total CPU seconds "time to end" to the end of runs. This extra time was spent to prove that we can not improve the best solution obtained so far by the algorithm. The overall ARPD for the SA algorithm is 3.27% as compared to 4.87% for the 2 + FI algorithm, but at a cost of an ACT of 3275 seconds as opposed to 2941 seconds, respectively. This presents a percentage improvement in solution quality for the SA algorithm of 48.92% at only an 11.35% percent increase in ACT. Furthermore, the SA algorithm generates a reduction in the total number of vehicles used and finds new reduced vehicle numbers for four problems, marked with [a] in table 3. The 2 + FI algorithm did not identify any reduction of this kind.

Next, the performances of the two TS algorithms, TS + FBA and TS + BA, are analyzed, following the implementation discussed earlier in section 4 using the cost evaluation procedure (c), a value of $5 \times n$ for the stopping parameter (*MAXI*), and different tabu list sizes ranging from $\lceil n/2 \rceil, \lceil n/3 \rceil, \ldots$ to $\lceil n/6 \rceil$. The best computational results for the TS + FBA and the TS + BA algorithms are reported in table 4. The TS + FBA algorithm provides thirteen better solutions and three equal to the previously best published solutions for seventeen test problems, identified with [b] and an asterisk *, respectively. It also finds the best solutions for six out of nine random test problems. The TS + FBA algorithm is robust and its ARPD values range from 0 to 1.96%. The TS + BA algorithm finds twelve better solutions and three equal to the best published solutions, also identified with [b] and an asterisk *, respectively. It finds the best solutions for only three random problems. The ARPD values vary from 0 to 2.67%.

Both TS algorithms find four new best solutions with reduced vehicle numbers identified with * in table 4. An average performance analysis demonstrates the superiority of the TS + FBA over the TS + BA algorithm with respect to solution

Table 3

Computational results for the (SA) simulated annealing algorithm.

Problem number	Best published	SA value	Best iteration	CPU to best	CPU to end
G1	875/4	**875**[*]/4	1817	12.1	107.0
CL	1214/7	**1213**/7	9918	44.7	58.0
G2	810/4	**810**[*]/4	310	2.8	6.0
C1	524/5	528/5	566	8.7	167.4
C2	855/10	**838**[b]/10	26010	3564.3	6434.3
C3	833/8	**829**[b]/8	59244	6171.2	9334.0
C4	1082/12	**1058**[b]/12	69609	4293.0	5012.3
C5	1351/17	1378/**16**[a]	34431	1373.8	2318.1
C6	560/6	**555**[b]/6	10427	697.8	3410.2
C7	916/12	**909**[b]/**11**[a]	17478	311.3	626.5
C8	885/9	**866**[b]/9	9810	364.2	957.2
C9	1210/15	**1164**[b]/**14**[a]	739981	59017.1	84301.2
C10	1464/19	**1417**[b]/**18**[a]	25648	2417.3	5708.0
C11	1046/7	1176/7	2252	266.2	315.8
C12	822/10	826/10	1516	48.7	632.0
C13	1551/11	**1545**[b]/11	39162	4569.2	7622.5
C14	874/11	890/11	6457	300.4	305.2
N1	709/6	757/6	3020	43.8	49.9
N2	814/8	864/8	8900	71.3	99.9
N3	994/10	1032/10	399	21.5	29.0
N4	925/7	1014/7	4327	122.7	125.1
N5	1045/9	1098/9	3074	145.2	150.9
N6	1011/11	1091/11	10664	102.5	166.0
N7	1035/8	1135/8	9910	513.9	549.0
N8	1185/10	1264/10	3403	319.1	363.9
N9	1234/12	1350/12	15052	350.7	353.3
ARPD[d]	–	3.37	–	3275.2[c]	4969.3[c]

[*] Best published solution value was obtained.
[a] Better number of vehicles was found by the SA algorithm.
[b] Better solution value than published was found by the SA algorithm.
[c] Average computation time in CPU seconds.
[d] ARPD: Average relative percentage deviation over the new best solutions.

quality with an ARPD of 0.43% and an ACT of 966 seconds, as opposed to 0.66% and an ACT of 499 seconds, respectively. This reduction in ACT is mainly due to the proposed special data structure.

Finally, an excellent regression fit was observed for eq. (5) with an R-squared value of 0.825. The estimated coefficient values are significant and different from zero at the 99% confidence level. A good fit was also obtained from eq. (4) with

Table 4

Computational results of the TS + FBA and TS + BA algorithms.

Problem number	Best published	TS + FBA results					TS + BA results				
		Solution value	Tabu size	Best iteration	CPU to best	CPU to end	Solution value	Tabu size	Best iteration	CPU to best	CPU to end
G1	875/4	875*/4	10	107	10.8	22.4	875*/4	10	75	5.7	16.9
CL	1214/7	1205[b]/7	22	1365	36.1	50.8	1210[b]/7	18	410	10.6	18.4
G2	810/4	819*/4	9	96	5.3	22.6	810*/4	9	24	2.1	13.4
C1	524/5	524*/5	13	529	61.4	114.0	524*/5	11	278	35.3	67.2
C2	855/19	844[b]/10	26	247	50.3	178.7	844[b]/10	26	190	23.8	70.8
C3	833/8	838/8	26	1260	894.6	1543.0	835/8	34	730	400.6	675.0
C4	1082/12	1044[b]/12	36	1373	1761.3	3560.0	1052[b]/12	38	3434	2488.1	3075.0
C5	1351/17	1334[b]/16[a]	34	895	1703.9	3246.0	1354/16[a]	40	3851	1542.2	1972.7
C6	560/6	555[b]/6	17	233	62.9	173.0	555[b]/6	13	381	84.6	140.2
C7	916/12	911[b]/11[a]	16	1654	744.6	1056.7	913[b]/11[a]	19	593	124.3	203.0
C8	885/9	878[b]/9	21	1641	1964.7	2998.0	866[b]/9	21	1075	819.0	1200.0
C9	1210/15	1184[b]/14[a]	51	895	2474.7	4755.8	1188[b]/14[a]	38	1196	1446.0	2443.6
C10	1464/19	1441[b]/18[a]	100	968	4024.6	4561.0	1422[b]/18[a]	34	1194	1726.6	3310.1
C11	1046/7	1043[b]/7	41	745	780.3	1445.4	1042[b]/7	31	858	803.4	1398.4
C12	822/10	819[b]/10	26	339	339.8	892.2	819[b]/10	21	249	127.0	407.5
C13	1551/11	1545[b]/11	61	821	1576.3	2834.0	1547[b]/11	31	551	613.5	1343.0
C14	874/11	866[b]/11	34	543	581.5	1175.9	866[b]/11	29	24	413.2	5579.0
N1	709/6	716/6	11	153	38.4	136.5	709[b]/6	9	56	11.4	62.5
N2	814/8	830/8	9	747	146.5	233.2	814[b]/8	17	752	101.3	135.0
N3	994/10	994[b]/10	26	501	87.6	160.8	1005/10	17	201	18.5	41.5
N4	925/7	925[b]/7	13	827	540.7	933.7	946/7	76	224	107.1	286.5
N5	1045/9	1066/9	19	1234	726.3	1100.7	1045[b]/9	19	620	200.0	321.0
N6	1011/11	1011[b]/11	26	718	339.3	638.5	1017/11	38	903	203.0	287.3
N7	1035/8	1035[b]/8	17	1762	2444.8	3636.6	1056/8	21	308	271.7	713.0
N8	1185/10	1185[b]/10	26	1742	1935.0	2877.0	1209/10	34	1472	888.1	1189.8
N9	1234/12	1234b/12	26	1882	1786.8	2592.0	1267/12	17	1143	509.1	731.8
ARPD[d]		0.42		–	966.1[c]	1574.5[c]	0.66			499.7[c]	992.4[c]

* Best published solution value was obtained.
[a] Better number of vehicles was found.
[b] Better solution value than published was found.
[c] Average computation time in CPU seconds.
[d] ARPD: Average relative percentage deviation over the new best solutions.

an R-squared value of 0.67 and a 99% confidence level for the estimated coefficient values. These estimates of the tabu list size values and the alteration scheme developed for it are used only for the large sized problems. The idea has emerged after an analysis on relatively small sized problems was made.

Table 5

Evaluation of metastrategy methods.

Problem Number	Size	C&W $C(S)$	C&W v	SA $C(S)$	SA v	TS + FBA $C(S)$	TS + FBA v	TS + BA $C(S)$	TS + BA v	Best published $C(S)$[a]	Best published v	New best solution $C(S_{best})$	New best solution v
G1	29	1017	5	875	4	875	4	875	4	875 [10]	4	874.99	4
CL	30	1258	7	1213	7	1205	7	1210	7	1214 [45]	7	1205.00	7
G2	32	888	5	810	4	810	4	810	4	810 [10]	4	810.13	4
C1	50	625	5	528	5	524	5	524	5	524 [17]	5	524.61	5
C2	75	1005	10	838	10	844	10	844	10	855 [3]	10	838.62	10
C3	100	982	8	829	8	838	8	835	8	833 [17]	8	829.18	8
C4	150	1299	12	1058	12	1044	12	1052	12	1082 [11]	12	1044.35	12
C5	199	1707	17	1376	16	1334	16	1354	16	1387 [3]	17	1334.16	16
C6	50	670	6	555	6	555	6	555	6	560 [17]	6	555.44	6
C7	75	989	12	909	11	911	11	913	11	916 [17]	12	909.68	11
C8	100	1055	10	866	9	878	9	866	9	885 [17]	9	866.75	9
C9	150	1383	15	1164	14	1184	14	1188	14	1210 [3]	15	1164.12	14
C10	199	1671	20	1418	18	1441	18	1422	18	1464 [3]	19	1417.85	18
C11	120	1291	7	1176	7	1043	7	1042	7	1046 [44]	7	1042.11	7
C12	100	939	10	826	10	819	10	819	10	822 [44]	10	819.59	10
C13	120	1646	11	15455	11	1545	11	1547	11	1551 [3]	11	1545.98	11
C14	100	952	11	890	11	866	11	866	11	874 [3]	11	866.35	11
ARPD[b]		26.65	(170)	1.29	(163)	0.36	(163)	0.38	(163)	1.45	(167)	0.00	(163)

[a] [·]: Numbers in brackets represent the reference in which the solution was found.

[b] (x): Shows the total number of vehicles used by the algorithm.

6. Comparative analysis and conclusions

In this study, we have developed λ-interchange descent methods for the vehicle routing problem and superimposed metastrategy simulated annealing and tabu search algorithms on the best of the descent methods. The objective is to compare their performance with respect to solution quality and computational time. We tested these approaches on classical routing problems with capacity and maximum distance constraints, and on randomly generated data with only capacity constraints. The results in table 5 are summarised for the seventeen test problems as follows:

(1) The constructive method of Clarke and Wright [14] produces solutions with an ARPD of 26.65% and a total number of 170 vehicles, which is about 4.3% away from the optimal solution. λ-interchange descent methods with ($\lambda = 1$ and $\lambda = 2$) improve substantially the C&W results in the case of the 1 + FI, 2 + FI and the 1 + BI algorithms. The best-improve with approximate cost and special data structure 1 + BI + DS reduced the average computation time

of the 1 + BI algorithm with a small sacrifice in solution quality. In general, descent methods fail to reduce the number of vehicles and produce the published results ($v = 167$).

(2) Simulated annealing produces new best solutions using a total of 163 vehicles, but displays large variance with regard to solution quality and computational time. The ARPD is 1.29% with an ACT of 4909 seconds to the best solutions.

(3) Both tabu search schemes with a first-best-admissible strategy (TS + FBA) and a best admissible strategy (TS + BA) outperform the SA algorithm in solution quality and computation time. Tabu search results are also more robust than SA. The TS + FBA algorithm produces an average relative percentage deviation (ARPD) of 0.36%, similar to the value of 0.38% for the TS + BA algorithm with an ACT of 1004 as opposed to 626 CPU seconds, respectively. The time reduction is due to the sophisticated data structure. Since the difference in the ARPDs is acceptable, the TS + BA algorithm seems to be a more efficient option when computer time is a scarce resource.

(4) Good estimates of a tabu list size and total number of iterations for tabu search schemes were found to depend on problem characteristics. An approach to vary the tabu list size around an interval was introduced to reduce the error in the estimate.

(5) The total number of vehicles obtained in the published literature (167) is larger than the new total of 163. Also the ARPD of published solutions is worse by 1.48% on average. Better solutions were found for fourteen out of the seventeen classical problems, and identical solutions were found in the three other cases, where these seem to be optimal. The largest improvements were obtained for problems of medium and large sizes, with or without time constraints, as in the case of problems C5, C10 (199 customers), where the ARPDs are 3.97%, 3.24% and the new vehicle numbers are 16, 18 rather than the published 17, 19, respectively. Due to this reduction, it is not necessary to confine oneself to a feasible starting solution if they are difficult to obtain. The metastrategy algorithms can also be applied to VRP with different vehicle sizes without any difficulties. We strongly recommend them to other related routing and distribution problems.

After the revision of this paper, we became aware of the work of Grandeau et al. [20]. They use a tabu search technique which performs tabu moves consisting of inserting cities into different routes. They allow infeasible moves to be considered during the search. This type of insertion resembles our shift process but without the interchange process. The algorithm also uses a post-optimization procedure to end the search. Computational results were provided for the C1–C14 problems. They obtain slightly better than our best solutions for three problems (C2–C4), equal solutions for four problems (C1, C6, C8, C14), and worse solutions for the six remaining problems. In the case of C5, they obtain a tour with a length of 1329.29

using 17 vehicles, while we obtain a tour of length 1334.16, compensated by using only 16 vehicles. It can be seen that our algorithms perform significantly better in the presence of time limits and the clustered problems. In addition, our results could be improved by a post-optimization procedure which is not used in this study.

Appendix

This appendix contains the best real solutions produced in this study for the seventeen test problems using real-valued distances. The input and output data for the nine random problems are not included due to the limited space, but can be obtained from the author. For every problem we present: the total route length value $C(S)$ includes the drop times (value in brackets, when applicable, does not) and the individual route length $C(R_p)$, the unused capacity $\overline{Q} = Q - \sum_{i \in R_p} d_i$; the number of customers, and the sequence of customers in each route.

Problem G1: $n = 29$, $C(S) = \mathbf{874.99}$, $Q = 4500$, $L = 240$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	233.95	1650	6	0-26-28-27-25-24-29-0
2	236.59	125	8	0-22-2-5-4-1-6-3-20-0
3	177.24	1700	5	0-23-8-14-21-19-0
4	227.21	1775	10	0-15-16-13-7-17-9-12-11-10-18-0

Problem CL: $n = 30$, $C(S) = \mathbf{1205.00}$, $Q = 140$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	177.0	0	4	0-4-29-23-0
2	219.00	0	9	0-12-26-9-7-15-16-11-13-0
3	86.00	64	4	0-2-1-17-0
4	214.0	1	7	0-24-3-28-6-5-22-0
5	168.00	1	5	0-20-25-8-19-0
6	138.00	8	3	0-21-30-0
7	203.00	3	5	0-14-27-10-18-0

Problem G2: $n = 32$, $C(S) = \mathbf{810.13}$, $Q = 80000$, $L = 240$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	227.75	150	10	0-18-19-21-20-22-23-24-25-17-14-0
2	177.88	80	10	0-13-32-10-9-8-7-6-5-11-1-0
3	181.82	750	6	0-29-28-16-27-26-15-0
4	222.68	1650	6	0-12-2-4-3-30-31-0

Problem C1: $n = 50$, $C(S) = 524.61$, $Q = 160$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	118.52	11	11	0-8-26-31-28-3-36-35-20-22-1-32-0
2	99.25	0	11	0-12-37-44-15-45-33-39-10-49-5-46-0
3	98.45	8	9	0-6-14-25-24-43-7-23-48-27-0
4	109.06	3	9	0-47-4-17-42-19-40-41-13-18-0
5	99.33	1	10	0-38-9-30-34-50-16-21-29-2-11-0

Problem C2: $n = 75$, $C(S) = 838.62$, $Q = 140$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	119.38	4	8	0-32-50-18-55-25-31-10-72-0
2	40.43	8	6	0-75-27-52-46-34-67-0
3	55.35	3	7	0-4-45-29-5-48-30-68-0
4	89.34	2	7	0-74-21-61-28-22-62-2-0
5	81.63	3	8	0-6-33-16-49-24-44-3-51-0
6	106.59	1	9	0-73-1-43-42-64-41-56-23-63-0
7	81.81	2	6	0-58-38-65-66-11-35-0
8	95.33	1	7	0-7-53-14-59-19-54-8-0
9	115.82	4	11	0-47-36-69-71-60-70-20-37-15-57-13-0
10	52.93	8	6	0-26-12-39-9-40-17-0

Problem C3: $n = 100$, $C(S) = 829.18$, $Q = 200$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	124.65	25	11	0-88-62-11-19-49-64-63-90-32-10-31-0
2	83.10	43	13	0-58-2-57-41-22-75-74-72-73-21-40-52-0
3	111.50	6	12	0-13-87-42-43-14-44-38-86-16-61-99-6-0
4	107.08	47	9	0-26-4-56-23-67-39-25-55-54-0
5	80.45	19	12	0-28-76-77-3-79-33-81-9-51-50-1-27-0
6	59.35	1	12	0-94-95-97-92-98-37-100-91-85-93-59-96-0
7	138.67	1	15	0-12-80-68-24-29-78-34-35-71-65-66-20-30-70-69-0
8	124.38	0	16	0-89-18-83-60-5-84-17-45-8-46-36-47-48-82-7-52-0

Problem C4: $n = 150$, $C(S) = 1044.35$, $Q = 200$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	123.50	0	14	0-132-122-20-66-71-65-136-35-135-120-9-103-51-1-0
2	114.67	5	13	0-89-5-84-17-113-86-140-38-43-15-57-144-87-0
3	128.71	0	15	0-146-7-123-19-107-11-64-49-143-36-47-124-46-114-18-0
4	109.85	8	12	0-149-54-130-55-25-139-39-67-23-56-4-110-0
5	94.52	0	16	0-52-88-148-62-10-108-126-63-90-32-131-128-30-70-101-69-0
6	77.53	0	10	0-106-82-48-8-45-125-83-60-118-147-0
7	73.06	3	13	0-105-26-109-134-24-29-121-68-80-150-12-138-28-0
8	87.12	1	15	0-13-117-97-42-142-14-119-44-141-16-61-91-100-37-96-0
9	76.22	0	13	0-111-50-102-33-81-34-78-129-79-3-77-116-76-0
10	72.66	6	15	0-58-137-2-115-145-41-22-133-75-74-72-73-21-40-53-0
11	35.70	142	3	0-27-31-127-0
12	50.81	0	11	0-6-99-104-59-93-85-98-92-95-94-112-0

Problem C5: $n = 199$, $C(S) = 2344.16$, $Q = 200$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	118.13	0	13	0-1-51-161-71-65-136-35-135-164-34-169-29-121-0
2	106.49	3	14	0-98-100-192-14-119-44-38-140-86-113-17-84-60-166-0
3	137.29	2	14	0-88-159-126-63-181-64-49-143-36-47-168-124-46-18-0
4	109.08	1	13	0-21-197-56-186-23-67-39-139-187-170-25-55-165-0
5	81.92	2	11	0-152-48-123-19-107-175-11-62-148-182-52-0
6	98.45	0	13	0-184-116-3-129-78-120-9-103-66-188-20-122-176-0
7	81.55	0	14	0-53-152-58-137-144-57-15-43-142-42-172-87-97-94-0
8	77.18	0	13	0-106-194-7-82-114-8-174-45-125-199-83-6-183-0
9	85.75	0	13	0-189-10-108-90-32-131-160-128-30-70-101-111-28-0
10	71.58	1	12	0-26-149-195-179-54-134-24-163-68-150-80-12-0
11	71.17	0	12	0-96-37-193-91-191-141-16-61-173-5-118-89-0
12	72.49	1	13	0-2-178-115-145-41-22-133-75-74-171-72-73-40-0
13	60.80	2	11	0-50-102-157-33-81-185-79-158-77-196-76-0
14	52.16	0	11	0-13-117-95-92-151-59-85-93-99-104-147-0
15	61.94	0	12	0-138-154-109-177-130-4-155-110-198-180-105-0
16	48.19	2	12	0-156-112-146-167-127-190-31-162-69-132-27-0

Problem C6: $n = 50$, $C(S) = 1055.44$ (555.44), $Q = 160$, $L = 200$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	198.08	23	9	0-2-20-35-36-3-28-31-22-1-0
2	189.94	29	8	0-17-42-19-40-41-13-25-14-0
3	195.33	19	10	0-32-11-16-29-21-50-34-30-9-38-0
4	82.33	80	4	0-18-4-47-46-0
5	199.12	5	10	0-12-37-44-15-45-33-39-10-49-5-0
6	190.64	27	9	0-6-23-24-43-7-26-8-48-27-0

Problem C7: $n = 75$, $C(S) = 1659.68$ (909.68), $Q = 140$, $L = 160$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	153.82	0	8	0-75-48-47-36-37-5-29-45-0
2	127.16	11	5	0-38-65-66-11-53-0
3	146.84	8	7	0-16-49-24-3-44-40-17-0
4	158.76	53	6	0-4-20-70-60-71-69-0
5	159.92	28	7	0-62-22-64-42-1-73-6-0
6	144.38	0	7	0-68-2-28-61-21-74-30-0
7	151.69	17	7	0-12-72-39-31-10-58-26-0
8	152.97	27	6	0-32-50-18-55-25-9-0
9	155.24	25	7	0-33-43-41-56-23-63-51-0
10	157.54	5	8	0-27-15-57-13-54-52-34-67-0
11	151.36	2	7	0-46-8-19-59-14-35-7-0

Problem C8: $n = 100$, $C(S) = 1866.75$ (866.75), $Q = 200$, $L = 230$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	200.12	45	11	0-27-69-70-30-32-90-63-10-62-88-31-0
2	227.55	22	11	0-18-82-48-47-36-49-64-11-19-7-52-0
3	197.08	47	9	0-54-55-25-39-67-23-56-4-26-0
4	221.37	7	11	0-13-87-42-43-14-44-38-86-16-61-96-0
5	227.93	37	11	0-50-33-81-9-35-71-65-66-20-51-1-0
6	178.60	110	9	0-89-60-83-8-46-45-17-84-5-0
7	213.10	43	13	0-53-40-21-73-72-74-75-22-41-15-57-2-59-0
8	190.74	0	13	0-94-95-97-92-37-98-100-91-85-93-59-99-6-0
9	210.26	31	12	0-12-80-68-24-29-34-78-79-3-77-76-28-0

Problem C9: $n = 150$, $C(S) = 2664.12$ (1164.12), $Q = 200$, $L = 200$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	191.33	51	11	0-31-108-90-32-131-128-20-30-70-101-69-0
2	136.01	30	9	0-147-96-104-99-93-98-59-94-112-0
3	188.33	71	11	0-137-87-144-57-15-43-142-42-97-117-13-0
4	184.60	58	12	0-58-2-115-145-41-22-133-74-73-21-40-53-0
5	196.13	7	10	0-105-110-4-139-39-67-23-56-75-72-0
6	197.72	4	12	0-50-102-33-81-120-9-103-51-122-1-132-27-0
7	199.13	92	8	0-78-34-135-35-136-65-71-66-0
8	192.73	85	8	0-10-126-63-64-49-143-36-47-0
9	198.40	25	11	0-89-83-114-8-125-45-46-124-48-82-18-0
10	199.64	12	13	0-28-150-80-68-121-29-129-79-3-77-116-76-111-0
11	194.18	42	12	0-127-88-148-62-11-107-19-123-7-106-52-146-0
12	193.96	1	11	0-60-118-5-84-17-113-86-141-16-61-6-0
13	192.39	46	11	0-138-12-109-134-24-25-55-130-54-149-26-0
14	199.57	41	11	0-95-92-37-100-119-14-38-140-44-91-85-0

Problem C10: $n = 199$, $C(S) = \mathbf{3407.85}$ (1417.85), $Q = 200$, $L = 200$, $\delta_i = 10$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	198.95	15	12	0-76-196-77-158-3-79-129-169-78-34-164-185-0
2	197.81	2	11	0-61-16-191-141-86-113-17-173-84-5-147-0
3	196.58	88	8	0-174-46-36-143-49-64-11-148-0
4	194.54	6	12	0-21-197-56-186-39-187-139-4-155-110-198-180-0
5	198.37	32	11	0-183-59-100-192-119-44-140-38-14-97-117-0
6	193.07	12	11	0-88-182-123-19-107-175-62-159-189-10-70-0
7	193.47	33	9	0-81-120-135-35-136-65-71-161-51-0
8	198.91	23	11	0-50-102-157-33-9-103-66-188-20-122-1-0
9	197.70	21	12	0-184-116-68-80-150-121-29-24-163-134-54-177-0
10	196.06	37	10	0-108-90-126-63-181-32-160-128-30-0
11	175.27	20	11	0-89-166-118-60-83-199-125-45-8-114-18-0
12	158.80	11	11	0-27-167-127-190-31-162-101-69-132-176-111-0
13	141.30	18	10	0-28-154-138-12-109-195-149-26-105-53-0
14	198.82	33	10	0-72-75-23-67-170-25-55-130-179-0
15	195.43	0	13	0-152-58-2-178-115-145-41-22-133-74-171-73-40-0
16	194.14	1	14	0-6-96-99-104-93-85-91-193-98-37-151-92-95-94-0
17	181.92	36	11	0-146-52-106-194-7-48-168-47-124-82-153-0
18	196.71	26	12	0-156-112-13-87-172-42-142-43-15-57-144-137-0

Problem C11: $n = 120$, $C(S) = \mathbf{1042.11}$ (555.44), $Q = 200$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	213.63	1	16	0-52-54-57-59-65-61-62-64-66-63-60-56-58-55-53-100-0
2	207.94	3	21	0-109-21-20-23-26-28-32-35-29-36-34-31-30-33-27-24-22-25-19-16-17-0
3	199.63	0	16	0-95-37-38-39-42-41-44-46-47-49-50-51-48-45-43-40-0
4	144.43	1	16	0-106-73-76-68-77-79-80-78-72-75-74-71-70-69-67-107-0
5	134.96	1	16	0-88-2-1-3-4-5-6-7-9-10-11-15-14-13-12-8-0
6	74.56	7	17	0-87-92-93-96-94-97-115-110-98-116-103-104-99-101-102-105-120-0
7	66.96	12	18	0-82-111-86-85-89-91-90-114-18-118-108-83-113-117-84-112-81-119-0

Problem C12: $n = 100$, $C(S) = \mathbf{819.59}$, $Q = 200$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	137.02	0	14	0-81-78-76-71-70-73-77-79-80-72-61-64-68-69-0
2	101.88	0	8	0-55-54-53-56-58-60-59-57-0
3	95.94	10	9	0-99-100-97-93-92-94-95-96-98-0
4	97.23	0	9	0-32-33-31-35-37-38-39-36-34-0
5	96.04	0	9	0-13-17-18-19-15-16-14-12-10-0
6	76.07	30	10	0-91-89-88-85-84-82-83-86-87-90-0
7	64.81	40	13	0-47-49-52-50-51-48-45-46-44-40-41-42-43-0
8	56.17	30	11	0-5-3-7-8-11-9-6-4-2-1-75-0
9	43.59	50	6	0-67-65-63-74-62-66-0
10	50.84	30	11	0-21-23-26-28-30-29-27-25-24-22-20-0

Problem C13: $n = 120$, $C(S) = 7545.98$ (1545.98), $Q = 200$, $L = 720$, $\delta_i = 50$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	705.09	86	10	0-53-55-58-56-60-63-66-64-62-57-0
2	690.47	78	10	0-37-44-46-47-49-50-51-48-43-40-0
3	691.61	128	10	0-21-26-32-35-31-30-33-34-36-29-0
4	715.87	53	10	0-52-54-61-65-59-45-42-41-38-39-0
5	672.87	88	10	0-17-16-19-25-22-24-27-28-23-20-0
6	686.49	59	11	0-68-76-77-79-80-78-75-72-74-71-73-0
7	717.85	45	12	0-108-8-12-13-14-15-11-10-9-7-84-85-0
8	711.72	73	12	0-101-99-100-116-110-98-67-70-69-103-104-107-0
9	706.69	66	12	0-112-117-113-83-6-5-4-3-1-2-81-119-0
10	708.88	51	13	0-96-93-94-97-115-109-114-118-18-90-91-89-92-0
11	538.44	98	10	0-120-105-106-102-95-87-86-111-82-88-0

Problem C14: $n = 100$, $C(S) = 9866.36$ 866.35), $Q = 200$, $L = 1040$, $\delta_i = 90$.

p	$C(R_p)$	$\overline{Q}$	$\lvert R_p \rvert$	Route
1	1028.04	0	10	0-63-80-79-77-73-70-71-76-78-81-0
2	821.88	0	8	0-55-54-53-56-58-60-59-57-0
3	996.70	0	10	0-98-96-95-94-92-93-97-100-99-1-0
4	907.23	0	9	0-32-33-31-35-37-38-39-36-34-0
5	906.04	0	9	0-10-12-14-16-15-19-18-17-13-0
6	976.07	30	10	0-91-89-88-85-84-82-83-86-87-90-0
7	871.56	90	9	0-47-46-45-48-51-50-52-49-20-0
8	957.79	50	10	0-67-65-62-74-72-61-64-68-66-69-0
9	949.41	40	10	0-21-22-24-25-27-29-30-28-26-23-0
10	956.17	40	10	0-5-3-7-8-11-9-6-4-2-75-0
11	495.47	140	5	0-43-42-44-40-41-0

Acknowledgements

This work was supported by the Hariri Foundation, Lebanon. Thanks are due to Professor Nicos Christofides for helpful discussions and to the referees for their useful comments.

References

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machine* (Wiley, 1989).

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

[3] K. Altinkemer and B. Gavish, Parallel savings based heuristics for the delivery problem, Oper. Res. 39(1991)456–469.

[4] J. Beasley, Route first-cluster second methods for vehicle routing, Omega 118(1983)403–408.
[5] W. Bell, L. Dalberto, M. Fisher, A. Greenfield, R. Jaikumar, R. Mack and P. Prutzman, Improving distribution of industrial gases with an on-line computerized routing and scheduling systems, Interfaces 13(1983)4–23.
[6] L. Bodin, B. Golden, A. Assad and M. Ball, Routing and scheduling of vehicles and crews: The state of the art, Comp. Oper. Res. 10(1983)69–211.
[7] L. Bodin, Twenty years of routing and scheduling, Oper. Res. 38(1990)571–579.
[8] G. Brown and G. Graves, Real-time dispatch of petroleum tank trunks, Manag. Sci. 27(1981) 19–32.
[9] N. Christofides, Vehicle routing, in: *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, ed. E. Lawler, J. Lenstra, A. Rinnooy Kan and D. Shmoys (Wiley, 1985).
[10] N. Christofides and S. Eilon, An algorithm for the vehicle dispatching problem, Oper. Res. Quart. 20(1969)309–318.
[11] N. Christofides, A. Mingozzi and P. Toth, The vehicle routing problem, in: *Combinatorial Optimization*, ed. N. Christofides, A. Mingozzi, P. Toth and C. Sandi (Wiley, 1979).
[12] N. Christofides, A. Mingozzi and P. Toth, Exact algorithms for the vehicle routing problem, based on spanning tree shortest path relaxation, Math. Progr. 20(1981)255–282.
[13] N. Christofides, A. Mingozzi and P. Toth, State space relaxation procedures for the computation of bounds to routing problems, Networks 11(1981)145–164.
[14] G. Clarke and J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Oper. Res. 12(1964)568–581.
[15] S. Evans and J. Norback, The impact of a decision-support system for vehicle routing in a food service supply situation, J. Oper. Res. Soc. 36(1985)467–472.
[16] M. Fisher, R. Greenfield, R. Jaikumar and J. Lester, A computerized vehicle routing application, Interfaces 1(1982)45–52.
[17] M. Fisher and R. Jaikumar, A generalised assignment heuristic for vehicle routing, Networks 11(1981)109–124.
[18] M. Fisher, Lagrangian optimization algorithms for vehicle routing problems, in: *Operational Reseach'87, IFORS, 1988*, ed. G.K. Rand (Elsevier Science/North-Holland, 1988).
[19] T. Gaskell, Bases for vehicle fleet scheduling, Oper. Res. Quart. 18(1967)367–384.
[20] 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, Canada (1991).
[21] B. Gillet and L. Miller, A heuristic algorithm for vehicle dispatches, Oper. Res. 24(1976)340–349.
[22] F. Glover, Future paths for integer programming and links to artificial intelligence, Comp. Oper. Res. 13(1986)533–549.
[23] F. Glover, Tabu search, Part I, ORSA J. Comput. 1(1989)190–206.
[24] F. Glover, Tabu search, Part II, ORSA J. Comput. 2(1990)4–32.
[25] F. Glover, Simple tabu thresholding in optimization, Graduate School of Business, University of Colorado, Boulder (May 1992).
[26] B. Golden and E. Watts, Computerized vehicle routing in the soft drink industry, Oper. Res. 35(1987) 6–17.
[27] B. Golden and A. Assad, *Vehicle Routing: Methods and Studies* (Elsevier Science/North-Holland, 1988).
[28] M. Haimovich and A.H.G. Rinnooy Kan, Bounds and heuristics for capacitated routing problems, Math. Oper. Res. 10(1985)527–542.
[29] D.S. Johnson, Local optimization and the traveling salesman problem, *Proc. 17th Int. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science (1990) pp. 446–461.
[30] S. Kirkpatrick, J.C.D. Gelott and M.P. Vecchi, Optimization by simulated annealing, Science 220(1983)671–680.

[31] G. Laporte, Y. Nobert and M. Desrochers, Optimal routing under capacity and distance restriction, Oper. Res. 33(1985)1050–1073.
[32] G. Laporte and Y. Nobert, Exact algorithms for the vehicle routing problem, Ann. Discr. Math. 31(1987)147–184.
[33] J. Lenstra and A. Rinnooy Kan, Complexity of vehicle routing and scheduling problems, Networks 11(1981)221–228.
[34] S. Lin, Computer solutions of the traveling salesman problem, Bell Syst. Comp. J. 44(1965) 2245–2269.
[35] S. Lin and B.W. Kernighan, An effective heuristic algorithm for the travelling salesman problem, Oper. Res. 21(1973)2245–2269.
[36] R.H. Mole and S.R. Jameson, A sequential route-building algorithm employing a generalised savings criterion, Oper. Res. Quart. 27(1976)503–511.
[37] M. Nelson, K. Nygard, J. Griffin and W. Shreve, Implementation techniques for the vehicle routing problem, Comp. Oper. Res. 12(1985)273–283.
[38] I. Or, Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking, Ph.D. Dissertation, Northwestern University, Evanston, IL (1976).
[39] I.H. Osman, Metastrategy simulated annealing and tabu search for combinatorial optimization problems, Ph.D. Dissertation, The Management School, Imperial College of Science and Medicine, University of London, London (1991).
[40] I.H. Osman, Heuristics for combinatorial optimization problems: development and new directions, *Proc. 1st Seminar on Information Technology and Applications*, Markfield Conference Centre, Leicester, UK (1991).
[41] I.H. Osman, A comparison of heuristics for the generalised assignment problem, Working Paper, University of Kent, Canterbury, UK (1990).
[42] I.H. Osman and N. Christofides, Simulated annealing and descent algorithms for capacitated clustering problems, presented as EURO-XI, Beograd, Yugoslavia (1989).
[43] I.H. Osman and C.N. Potts, Simulated annealing for permutation flow-shop scheduling, Omega 17(1989)551–557.
[44] H. Paessens, Saving algorithms for the vehicle routing problem, Eur. J. Oper. Res. 34(1988) 336–344.
[45] R.A. Russell, An effective heuristic for the M-tour traveling salesman problem with some side conditions, Oper. Res. 25(1977)517–524.
[46] W.R. Stewart, Jr. and B.L. Golden, A Lagrangian relaxation heuristic for vehicle routing, Eur. J. Oper. Res. 15(1984)84–88.
[47] E. Taillard, Robust tabu search for the quadratic assignment problem, Working Paper ORWP 90/10, Département de Mathématiques, Ecole Polytechnic Fédérale de Lausanne, Switzerland (1990).