ELSEVIER

Artificial Intelligence in Engineering 15 (2001) 281–295

ARTIFICIAL INTELLIGENCE *in* ENGINEERING

www.elsevier.com/locate/aieng

Heuristic methods for vehicle routing problem with time windows

K.C. Tan[a,*], L.H. Lee[b], Q.L. Zhu[a], K. Ou[a]

[a]*Department of Electrical and Computer Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260*
[b]*Department of Industrial and Systems Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260*

Received 7 September 2000; accepted 20 December 2000

Abstract

This paper documents our investigation into various heuristic methods to solve the vehicle routing problem with time windows (VRPTW) to near optimal solutions. The objective of the VRPTW is to serve a number of customers within predefined time windows at minimum cost (in terms of distance travelled), without violating the capacity and total trip time constraints for each vehicle. Combinatorial optimisation problems of this kind are non-polynomial-hard (NP-hard) and are best solved by heuristics. The heuristics we are exploring here are mainly third-generation artificial intelligent (AI) algorithms, namely simulated annealing (SA), Tabu search (TS) and genetic algorithm (GA). Based on the original SA theory proposed by Kirkpatrick and the work by Thangiah, we update the cooling scheme and develop a fast and efficient SA heuristic. One of the variants of Glover's TS, strict Tabu, is evaluated and first used for VRPTW, with the help of both recency and frequency measures. Our GA implementation, unlike Thangiah's genetic sectoring heuristic, uses intuitive integer string representation and incorporates several new crossover operations and other advanced techniques such as hybrid hill-climbing and adaptive mutation scheme. We applied each of the heuristics developed to Solomon's 56 VRPTW 100-customer instances, and yielded 18 solutions better than or equivalent to the best solution ever published for these problems. This paper is also among the first to document the implementation of all the three advanced AI methods for VRPTW, together with their comprehensive results. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Vehicle routing problem; Time windows; Combinatorial optimisation; Heuristics; Simulated annealing; Tabu search; Genetic algorithm

1. Introduction

Logistics may be defined as 'the provision of goods and services from a supply point to various demand points' [2]. A complete logistic system involves transporting raw materials from a number of suppliers or vendors, delivering them to the factory plant for manufacturing or processing, movement of the products to various warehouses or depots and eventually distribution to customers. Both the supply and distribution procedures require effective transportation management. Good transportation management can practically save a private company a considerable portion of its total distribution cost. Potential cost savings constitute: lowered trucking cost due to more optimal routes and shorter distances, reduced in-house space and related costs, less penalty incurred due to untimely delivery. One of the most significant measures of transportation management is effective vehicle routing. Optimising of routes for vehicles given various constraints is the origin of vehicle routing problems (VRPs).

Fig. 1 describes a typical VRP. The solution includes two routes: Depot → 7 → 8 → 9 → 11 → 12 → Depot; Depot → 2 → 3 → 1 → 4 → 5 → 6 → 10 → Depot. Sometimes the depot is denoted as 0. The vehicle routing problem with time windows (VRPTW) is a well-known non-polynomial-hard (NP-hard) problem, which is an extension of normal VRPs, encountered very frequently in making decisions about the distribution of goods and services. The problem involves a fleet of vehicles set off from a depot to serve a number of customers, at different geographic locations, with various demands and within specific time windows before returning to the depot. The objective of the problem is to find routes for the vehicles to serve all the customers at a minimal cost (in terms of travel distance, etc.) without violating the capacity and travel time constraints of the vehicles and the time window constraints set by the customers. To date, there is no consistent optimising algorithm that solves the problem exactly using mathematical programming. Instead, many heuristic methods have been designed to solve VRPTW to near optima.

In Marshall Fisher's survey [4], he categorised vehicle routing methods into three generations. The first generation was simple heuristics developed in the 1960s and 1970s, which were mainly based on local search or sweep.

* Corresponding author.

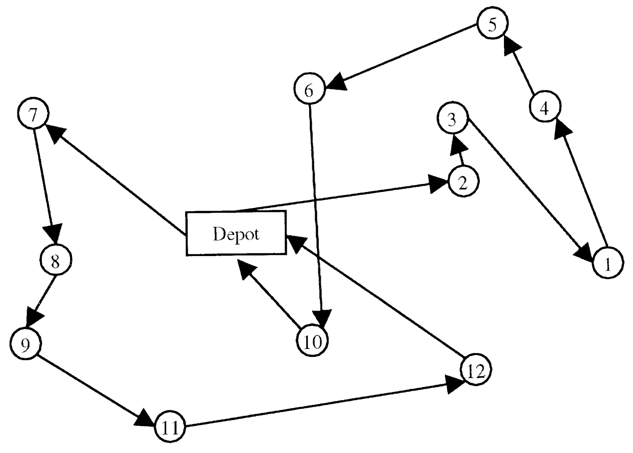

Fig. 1. A vehicle routing problem: a single depot VRP with 12 customers. Each route starts from depot, visiting customers and ends at depot.

Since these earlier studies were not well documented, it is hard to compare the results they obtained 30 years ago with the more recent solutions. The second generation, mathematical programming based heuristics, were near-optimisation algorithms that are very different from normal heuristics. These include the generalised assignment problems and set partitioning to approximate the VRP. Their results are usually superior to that of simple heuristics [4,20]. In fact for linear objective functions, some of these techniques are able to stretch to the optima. The third generation, or the one that is currently undergoing heavy research is exact optimisation algorithms and artificial intelligence methods. Among these, the most successful optimisation algorithms are K-tree, Lagrangian relaxation, etc., while the top AI representatives in VRPTW are simulated annealing (SA), Tabu search (TS) and genetic algorithms (GAs). These algorithms are discussed briefly as follows:

Kolen et al. [10] presented the method *of branch and bound*, which is among the first optimisation algorithms for VRPTW. The method calculates lower bounds using dynamic programming and state space relaxation. Branching decisions are taken on route-customer allocations. The method has successfully solved the problem involving 15 customers. Fisher [3] introduces an optimisation algorithm in which lower bounds are obtained from a relaxation based on a generalisation of spanning trees called *K-trees*. Capacity constraints are handled by introducing a constraint requiring that some set S, $S \subset C$, of the set of customers must be served by at least $k(S)$ vehicles. This constraint is Lagrangian relaxed and the resulting problem is still a K-tree problem with modified arc costs. Time window constraints are treated similarly. A constraint, requiring that not all arcs in a time violating path can be used, is generated and Lagrangian relaxed. The method has solved some of the 100-customer Solomon benchmark problems [18].

One of the effective approaches at present is the *shortest path composition*. The fundamental observation is, the only constraint which 'links' the vehicles together is that each customer in the network must be visited only once. The problem that consists of the rest of the constraints is an elementary shortest path problem with time windows and capacity constraints (ESPPTWCC) for each vehicle. Although this problem is strictly NP-hard, there are a few efficient dynamic programming algorithms for the slightly relaxed programs. Two decompositions have been investigated computationally, namely *Dantzig–Wolfe decomposition* and *variable splitting*. Desrochers et al. [25] implemented Dantzig–Wolfe decomposition, and solved up to some of the 100-customer Solomon benchmark problems. Researchers at Technical University of Denmark [9], on the other hand, suggested using variable splitting to solve the VRPTW with similar performance.

Thangiah et al. [21] developed a *λ-interchange local search descent* (*LSD*) *method* that uses a systematic insertion and swapping of customers between routes, defined as λ-interchange operators. Due to computation burden, only 1-interchange and 2-interchange are commonly used, which allows up to one or two customers to be inserted or swapped at one time. Although it is a fast algorithm, the performance is poor without the help from other heuristics. SA, first proposed by Kirkpatrick [8], searches the solution space by simulating the annealing process in metallurgy. The algorithm jumps to distant location in the search space initially. The step of the jumps is reduced as time goes on or as the temperature 'cools'. Eventually, the process will turn into a LSD method. Osman [14] has applied SA to solve the VRP by moving one customer from one route to another or exchanging two customers from two routes. TS is a memory-based search strategy that chooses the best solution contained in $N(S)$ that does not violate certain restrictions that prevent cycling. Usually, these restrictions are stored as queues in a structure called a *Tabu list*. Typical restrictions prevent making a move that has been done within the last t iterations, and a solution that has been encountered in the last t iterations is usually forbidden as well. TS stops after a fixed number of iterations. Gerdreau et al. applied TS using a neighbourhood that can be constructed by moving a single customer from one route to another. Osman and Talliard [14] used a neighbourhood that consists of all solutions obtained from inserting a customer and swapping two customers.

Holland developed the GA [7] method that codes the VRPTW solutions in forms of bit strings or chromosomes. The method starts with a population of random chromosomes. Fitter chromosomes are then selected to undergo a crossover and mutation process, as to produce children which are different from the parents but inherit certain genetic traits from the parents. This process is continued until a fixed number of generations has been reached or the evolution has converged. Thangiah [22] devised a genetic sectoring heuristic with special genetic representation that keeps the polar angle offset in the genes. The algorithm follows a cluster-first, route-second philosophy and solved 100-customer Solomon problems to near optima. Prinetto et al. [16] proposed a hybrid GA for the travelling salesman problem (TSP) in which 2-*opt* and *Or-opt* were incorporated with the GA. Blanton and Wainwright [1] presented two new crossover operators, merge cross #1 and merge cross #2, which are superior to traditional crossover operators. Shaw [17] presented *large neighbourhood search* (LNS), a method in *constraint programming*, to solve VRPTW. Relatedness plays a very important part in the selection of customer to remove and re-insert into the configuration using a constraint-based tree search. Shaw applied *limited discrepancy search* during the tree search to re-insert visits. The results were competitive to those obtained using operations research meta-heuristics.

In this paper, we further investigate and develop various advanced AI techniques including SA, TS and GA to effectively solve the VRPTW to near optimal solutions. Based on the original SA theory proposed by Kirkpatrick [8] and the work by Thangiah [21], we update the cooling scheme and develop a fast and efficient SA heuristic. One of the variants of Glover's TS, strict Tabu, is evaluated and first used for VRPTW, with the help of both recency and frequency measures. Our GA implementation, unlike Thangiah's genetic sectoring heuristic [21], uses an intuitive integer string representation and incorporates several new crossover operations and other advanced techniques such as hybrid hill-climbing and adaptive mutation scheme. We have tested our heuristics with all 56 Solomon's VRPTW instances and obtained complete results for these problem sets. There are totally four heuristics tested on the instances: 2-interchange method, SA, Tabu and GA. Their average performances are compared with the best-known solutions in the literature. From the result analysis, our TS and GA are already close to the best ways of solving VRPTW. Totally, we found 18 solutions better than or equivalent to the best-known results. The discussion of results is given in Section 8. In this paper, we give a mathematical model of VRPTW, followed by the design and implementation of the heuristics. The computational results are presented and discussed in the final part of the paper.

2. Problem formulation

This section describes the notation and features that are common through this paper. The VRPTW constraints consist of a set of identical vehicles, a central depot node, a set of customer nodes and a network connecting the depot and customers. There are $N + 1$ customers and K vehicles. The depot node is denoted as customer 0. Each arc in the network represents a connection between two nodes and also indicates the direction it travels. Each route starts from the depot, visits customer nodes and then returns to the depot. The number of routes in the network is equal to the number of vehicles used. One vehicle is dedicated to one route. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network.

In Solomon's 56 VRPTW 100-customer instances, all distances are represented by Euclidean distance, and the speed of all vehicles is assumed to be unity. That is, it takes one unit of time to travel one unit of distance. This assumption makes the problem simpler, because numerically the travel cost c_{ij}, the travel time t_{ij} and the Euclidean distance between the customer nodes equal each other.

Each customer in the network can be visited only once by one of the vehicles. Every vehicle has the same capacity q_k and each customer has a varying demand m_i. q_k must be greater or equal to the summation of all demands on the route travelled by vehicle k, which means that no vehicles can be overloaded. The time window constraint is denoted by a predefined time interval, given an earliest arrival time and latest arrival time. The vehicles must arrive at the customers not later than the latest arrival time, if vehicles arrive earlier than the earliest arrival time, waiting occurs. Each customer also imposes a service time to the route, taking consideration of the loading/unloading time of goods. In Solomon's instances, the service time is assumed to be unique regardless of the load quantity needed to be handled. Vehicles are also supposed to complete their individual routes within a total route time, which is essentially the time window of the depot.

There are three types of principal decision variables in VRPTW. The principal decision variable x_{ijk} ($i,j \in \{0,1,2,\ldots,N\}$; $k \in \{1,2,\ldots,K\}$; $i \neq j$) is 1 if vehicle k travels from customer i to customer j, and 0 otherwise. The decision variable t_i denotes the time when a vehicle arrives at the customer, and w_i denotes the waiting time at node i. The objective is to design a network that satisfies all constraints, at the same time minimising the total travel cost. The model is mathematically formulated below:

Principal decision variables:

t_i	arrival time at node i
w_i	wait time at node i

$x_{ijk} \in \{0,1\}$, 0 if there is no arc from node i to node j, and 1 otherwise. $i \neq j$; $i,j \in \{0,1,2,\ldots,N\}$.

Parameters:

K	total number of vehicles
N	total number of customers
y_i	any arbitrary real number
d_{ij}	Euclidean distance between node i and node j

c_{ij}	cost incurred on arc from node i to j
t_{ij}	travel time between node i and j
m_i	demand at node i
q_k	capacity of vehicle k
e_i	earliest arrival time at node i
l_i	latest arrival time at node i
f_i	service time at node i
r_k	maximum route time allowed for vehicle k

$$\text{Minimise} \quad \sum_{i=0}^{N} \sum_{j=0,j\neq i}^{N} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{1}$$

subject to:

$$\sum_{k=1}^{K} \sum_{j=1}^{N} x_{ijk} \le K \qquad \text{for } i = 0 \tag{2}$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik} \le 1 \qquad \text{for } i = 0 \qquad \text{and} \quad k \in \{1,\dots,K\} \tag{3}$$

$$\sum_{k=1}^{K} \sum_{j=0,j\neq i}^{N} x_{ijk} = 1 \qquad \text{for } i \in \{1,\dots,N\} \tag{4}$$

$$\sum_{k=1}^{K} \sum_{i=0,i\neq j}^{N} x_{ijk} = 1 \qquad \text{for } j \in \{1,\dots,N\} \tag{5}$$

$$\sum_{i=1}^{N} m_i \sum_{j=0,j\neq i}^{N} x_{ijk} \le q_k \qquad \text{for } k \in \{1,\dots,K\} \tag{6}$$

$$\sum_{i=0}^{N} \sum_{j=0,j\neq i}^{N} x_{ijk}(t_{ij} + f_i + w_i) \le r_k \qquad \text{for } k \in \{1,\dots,K\} \tag{7}$$

$$t_0 = w_0 = f_0 = 0 \tag{8}$$

$$\sum_{k=1}^{K} \sum_{i=0,i\neq j}^{N} x_{ijk}(t_i + t_{ij} + f_i + w_i) \le t_j \quad \text{for } j \in \{1,\dots,N\} \tag{9}$$

$$e_i \le (t_i + w_i) \le l_i \qquad \text{for } i \in \{1,\dots,N\} \tag{10}$$

Formula (1) is the objective function of the problem. Constraint (2) specifies there are maximum K routes going out of the depot. Eq. (3) makes sure every route starts and ends at the central depot. Eqs. (4) and (5) define that every customer node can be visited only once by one vehicle. Eq. (6) is the capacity constraint. Eq. (7) is the maximum travel time constraint. Constraints (8)–(10) define the time windows. These formulas completely specify the feasible solutions for VRPTW.

3. An initial solution

Most heuristic search strategies involve finding an initial feasible solution and then improving on that solution using local or global optimisation techniques. Here, we make use of the *push forward insertion heuristic* (PFIH), first introduced by Solomon [18] in 1987 as a method to create an initial route configuration. PFIH is an efficient method to insert customers into new routes.

The procedure is easy and straightforward. The method tries to insert the customer between all the edges in the current route. It selects the edge that has the lowest additional insertion cost. The feasibility check tests all the constraints including time windows and load capacity. Only feasible insertions will be accepted. When the current route is full, PFIH will start a new route and repeat the procedure until all the customers are routed. Usually, PFIH gives a reasonably good feasible solution in terms of the number of vehicles used. This initial number of vehicles provides an upper bound for the number of routes in the solution.

PFIH serves the role of constructing route configuration for VRPTW. It is an efficient method to obtain feasible solutions. The detail information can be obtained from Solomon's paper [18].

4. Local search with λ-interchange

The effectiveness of any iterative local search method is determined by the efficiency of the generation mechanism and the way the neighbourhood is searched. A *λ-interchange* generation mechanism was introduced by Osman and Christofides [13] for the capacitated clustering problem. It is based on customer interchange between sets of vehicle routes and has been successfully implemented with a special data structure to other problems by Osman [14], Thangiah [20], etc.

The local search procedure is conducted by interchanging customer nodes between routes. For a chosen pair of routes, the searching order for the customers to be interchanged needs to be defined, either systematically or randomly. In this paper, we only consider the cases $\lambda = 2$, which means that maximum two customer nodes may be interchanged between routes. Based on the number of λ, there are totally eight interchange operators are defined: (0,1), (1,0), (1,1), (0,2), (2,0), (2,1), (1,2), (2,2). The operator (1,2) on a route pair (R_p, R_q) indicates a shift of two customers from R_q to R_p and a shift of one customer from R_p to R_q. The other operators are defined similarly. For a given operator, the customers are considered sequentially along the routes. In both the shift and interchange process, only improved solutions are accepted if the move results in the reduction of the total cost.

There are two strategies to select between candidate solutions:

1. The first-best (FB) strategy will select the first solution in $N_\lambda(S)$, the neighbourhood of the current solution, that results in a decrease in cost.
2. The global-best (GB) strategy will search all solutions in $N_\lambda(S)$, where $N_\lambda(S)$ means the neighbourhood of current solution under λ-interchange operation. GB will select the one, which will result in the maximum decrease in cost.

In the following we describe the *λ-interchange LSD method*. LSD starts from an initial feasible solution obtained by the PFIH. The PFIH solution is further improved using the λ-interchange mechanism for a given number of iterations. The procedure of the λ-interchange LSD is shown below.

Algorithm 1. Local search descent method

```
LSD-1: Obtain a feasible solution $S$ for the VRPTW using
the PFIH.
LSD-2: Select a solution $S' \in N_\lambda(S)$.
LSD-3: If $\{C(S') < C(S)\}$, then
  accept $S'$ and go to LSD-2,
  else go to LSD-4.
LSD-4: If {neighbourhood of $N_\lambda(S)$ has been completely
searched: there are no moves
  that will result in a lower cost} then go to LSD-5
  else go to LSD-2.
LSD-5: Stop with the LSD solution.
```

The LSD result is dependent on the initial feasible solution. GB usually achieves better results than FB because it keeps track of all the improving moves but incurs more expensive computation time. On the other hand, LSD–FB is a blind search that accepts the FB result. In this paper, we implemented 2-interchange GB.

5. Simulated annealing

SA is a stochastic relaxation technique that finds its origin in statistical mechanics [11]. The SA methodology is analogous to the annealing processing of solids. In order to avoid the meta-stable states produced by quenching, metals are often cooled very slowly, which allows them time to order themselves into stable, structurally strong, low energy configurations. This process is called annealing. This analogy can be used in combinatorial optimisation with the states of the solids corresponding to the feasible solution, the energy at each state to the improvement in objective function and the minimum energy being the optimal solution [8]. SA involves a process in which the temperature is gradually reduced during the simulation. Often, the system is first heated and then cooled. Thus, the system is given the opportunity to surmount energetic barriers in a search for conformations with energies lower than the local-minimum energy found by energy minimisation. Unlike λ-interchange, SA is a global optimisation heuristic based on probability, therefore, is able to overcome local optima.

At each step of the simulation algorithm, a new state of the system is constructed from the current state by giving a random displacement to a randomly selected particle. If the energy associated with this new state was lower than the energy of the current state, the displacement was accepted, that is, the new state becomes the current state. If the new state had an energy higher by d joules, the probability of changing the current state to the new state is

$$\exp\left(-\frac{d}{kT}\right) \tag{11}$$

where k is the *Boltzmann constant* and T the absolute temperature at present. This basic step, a *metropolis step*, can be repeated indefinitely. The procedure is called a *metropolis loop*. It can be shown that this method of generating current states led to a distribution of states in which the probability of a given state with energy e_i to be the current state is

$$\frac{\exp(-e_i/kT)}{\sum_j \exp(-e_j/kT)} \tag{12}$$

This probability function is known as *Boltzmann density*. One of its characteristics is that for very high temperatures, each state has almost equal chances of being the current state. At low temperatures, only states with low energies have a high probability of being the current state. These probabilities are derived for a never ending executing of the metropolis loop. The advantages of this scheme is:

- SA can deal with arbitrary systems and cost functions;
- SA statistically guarantees finding an optimal solution;
- SA is relatively easy to code, even for complex problems;
- SA generally gives a ‘good’ solution.

However this original version of SA has some drawbacks:

- Repeatedly annealing with a $1/\log k$ schedule is very slow, especially if the cost function is expensive to compute.
- For problems where the energy landscape is smooth, or there are few local minima, SA is an overkill — simpler, faster methods (e.g. local descent) will work better. But usually one does not know what the energy landscape is.
- Normal heuristic methods, which are problem-specific or take advantage of extra information about the system, will often be better than general methods. But SA is often comparable to heuristics.
- The method cannot tell whether it has found an optimal

solution. Some other method (e.g. branch and bound) is required to do this.

In our modified version of SA, the algorithm starts with a relatively good solution resulting from PFIH. Initial temperature is set at $T_s = 100$, and is slowly decreased by

$$T_k = \frac{T_{k-1}}{1 + \tau\sqrt{T_{k-1}}} \tag{13}$$

where T_k is the current temperature at iteration k and τ a small time constant. The square root of T_k is introduced in the denominator to speed the cooling process. Here, we use a simple monotonically decreasing function to replace the $1/(\log k)$ scheme. Our scheme gives fairly good results in much less time. The algorithm attempts solutions in the neighbourhood of the current solution randomly or systematically and calculates the probability of moving to those solutions according to

$$P\,(\text{accepting a move}) = \exp\left(-\frac{\Delta}{T_k}\right) \tag{14}$$

This is a modified version of Eq. (11), where $\Delta = C(S') - C(S)$, $C(S)$ is the cost of the current solution and $C(S')$ the cost of the new solution. If $\Delta < 0$, the move is always warranted. One can see that as temperature cools down, the probability of accepting a non-cost-saving move gets exponentially smaller. When the temperature has gone to the final temperature $T_f = 0.001$ or there are no more feasible moves in the neighbourhood, we reset the temperature to

$$T_r = \max\left(\frac{T_r}{2}, T_b\right) \tag{15}$$

where T_r is the reset temperature, and was originally set to T_s, and T_b the temperature at which the best current solution was found. Final temperature is not set at zero because as temperature decreases to infinitesimally close to zero, there is virtually zero probability of accepting a non-improving move. Thus, a final temperature not equal but close to zero is more realistic. To search a local neighbourhood, the 2-interchange approach was adopted. Every time a GB solution is found, a 2-interchange (GB) procedure is executed to search for possible better solutions around it. The procedure terminates after a number of resets. Below is the detailed procedure of one of the SA implementations, which adopts a partial 2-interchange (FB) to search the neighbourhood.

T_s	starting temperature of the SA method = 100
T_f	final temperature of the SA method = 0.001
T_b	temperature at which the current best solution was found
T_r	reset temperature of the SA method, originally equal to T_s
T_k	temperature of the current solution
S	current solution
S_b	current best solution
R	number of resets to be done
τ	the time constant in the range of (0, 1).

Algorithm 2. Simulated annealing

```
Step SA-1: Obtain a feasible solution for the VRPTW using the PFIH.
Step SA-2: Improve S using the 2-interchange LSD with GB strategy.
Step SA-3: Set cooling parameters: $T_s = T_b = T_r = T_k = 100$, $\tau = 0.5$.
Step SA-4: Generate systematically an $S' \in N_2(S)$ by (2, 0) and (1, 0) operations, and compute $\Delta = C(S') - C(S)$, where $N_2(S)$ is the neighbourhood of current solution under 2-interchange operation, $C(S)$ and $C(S')$ means the cost of current solution and the newly generated solution, respectively.
Step SA-5: If {($\Delta \le 0$) or ($\Delta > 0$ and $\exp(-\Delta/T_k) \ge \theta$), where $\theta$ is a random number between [0, 1]} then
    set $S = S'$.
    if {$C(S) < C(S_b)$} then
    improve S using 2-interchange LSD (GB).
    update $S_b = S$ and $T_b = T_k$.
Step SA-6: Set $k = k + 1$. Update the temperature using Eq. (13).
    If {$N_2(S)$ is searched without any accepted move} then
    reset $T_r = max(T_r/2, T_b)$, and set $T_k = T_r$.
Step SA-7: if {R resets have been made since the last $S_b$ was found} then
    go to Step SA-8.
    else go to Step SA-4.
Step SA-8: Terminate SA and print results.
```

In general, our SA implementation is a simple and fast algorithm that solves many VRPTWs to near optima. Due to the GB approach in local neighbourhood search, the algorithm is able to result in stable local optimal solutions almost at all times. This is especially true if the global optimum in a problem is located very distant to the corresponding PFIH initial solutions. In that case SA may not have enough energy to traverse that far, given the limited number of temperature resets.

6. Tabu search

TS is a memory-based search strategy, originally proposed by Glover [6], to guide the local search method to continue its search beyond a local optimum. The algorithm keeps a list of moves or solutions that have been made or visited in the past. This list, known as a *Tabu list*, is a queue of fixed or variable size. The purpose of the Tabu list is to record a number of most recent moves and prohibit any repetition or cycling. The memory can be *recency* or *frequency* based. In case of recency-based memory, also known as short-term memory, the Tabu list of size N records the last N moves or configurations the algorithm has encountered and sets them as 'Tabu'. Frequency-based memory, also known as long-term memory, complements the recency-based memory by providing the additional information of how many times the Tabu moves or Tabu solutions have been attempted. Frequency-based memory naturally provides better incentive as to the choice of next move. Despite that some of the moves are taboos, they can still materialise if they meet certain *aspiration criteria*. One obvious criterion to use is if the move results in a GB, it should be adopted even if it has been made very recently or very frequently.

Although a number of researchers, such as Willard [24], Osman [14], Gendreau et al. [5] and Taillard [19], have applied TS on VRP, few have attempted VRPTW. Potvin et al. [15] described a TS algorithm applicable to VRPTW but their primary objective was to minimise the number of vehicles instead of the total cost (distance), and their approach made limited use of the Tabu memory. Our design of TS for VRPTW combines short-term recency memory and long-term frequency memory and makes full use of intensification and diversification strategies. We created a special multi-functional list structure that can serve as a first-in–first-out queue structure or an array of descending or ascending order to store route configurations and move structures.

6.1. The Tabu list

When this list structure is used as a queue, it is a *Tabu list* that stores two kinds of recency information. One of them is the recently made moves. The moves have the structure:

$$\langle R_1, \text{node1}, \text{position1}, R_2, \text{node2}, \text{position2}\rangle \tag{16}$$

where R_1 and R_2 are the two routes under operation, node1 a node from R_1 and position1 the original position of node1. Likewise for node2 and position2. This notion provides a guideline to avoid making similar moves in the near future. Such representation does not uniquely describe a move, because a full description is very complicated and its use increases the computation tremendously. Another recency information stored in Tabu list is the solution configurations. Every solution that has been recently encountered is coded into an integer string. For example, if we have the following solution as described in Fig. 1:

Route No. 1: $0 \to 7 \to 8 \to 9 \to 11 \to 12 \to 0$
Route No. 2: $0 \to 2 \to 3 \to 1 \to 4 \to 5 \to 6 \to 10 \to 0$

The coded integer string is then

$$\langle 0 \to 7 \to 8 \to 9 \to 11 \to 12 \to 0 \to 2 \to 3 \to 1 \to 4 \to 5 \to 6 \to 10 \to 0\rangle \tag{17}$$

The zeros delimit the routes. This representation has 1-to-1 correspondence to the solution itself. We also attach the total cost of this solution to the string, for reasons explained in the next section. The lifetime of the Tabu moves and Tabu solutions on the Tabu list is governed by the *Tabu list size* (TLS). The larger the TLS, the longer these moves and solutions remain as Tabu. Nevertheless, we found that longer TLS may restrict the search process from proceeding and cause it to end prematurely. Hence in most occasions, we fix the TLS at 8–10. The Tabu list also employs a frequency measure that counts the number of times each of the Tabu moves and solutions have been attempted by the search. This frequency information is important in determining the status of the search process. High Tabu-hit frequency implies the search has been caught in a local optimum and search ought to be stopped.

6.2. The candidate list

The long-term memory approach in our TS algorithm is best represented in the *candidate list*. This list stores the *elite solutions* the system has discovered in the search process, also in integer string form like Eq. (17), but ranked according to the total cost attached to the string, instead of a FIFO queue. An item on the candidate list remains there as long as it has not been intensified. Once in a while, some of these elite solutions may be visited by other paths and become Tabu. But since their lifetime as elite candidates are usually longer than that as a Tabu, most of these solutions can still be utilised during intensification. The size of the candidate list can be variable, but is generally a multiple of the size of the Tabu list.

6.3. Intensification and diversification

Our TS procedure begins with an initial solution obtained from PFIH. Assuming that the initial solution is good and close to the ultimate optimal point, it is intensified by undergoing a 2-interchange (GB) LSD procedure. As the current solution is being updated, the moves involved and the solution itself are also being copied to the Tabu list to prevent cycles or duplications. If the current solution is a new GB, it is also copied to the elite list for future exploration. The intensification terminates when the entire $N_\lambda(S)$ has been searched without any improved solutions or the maximum Tabu-hit frequency has been reached.

After one round of intensification, the current solution S is believed to have reached the optimum of its neighbourhood,

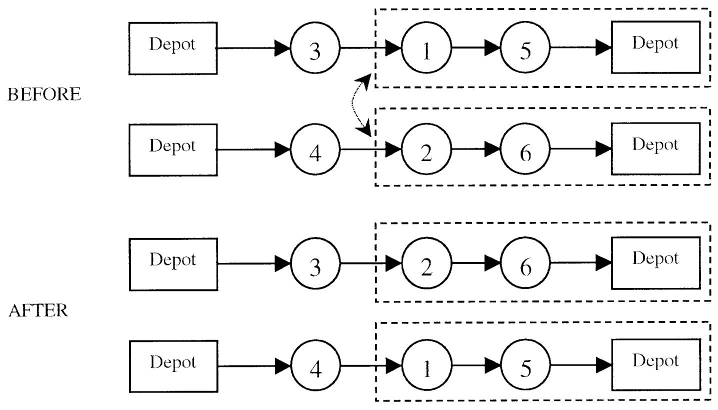

Fig. 2. A relink operation: relink operation simply exchanges customer nodes between routes in order to diversify the solution. Feasibility check must be performed.

and now it is time to diversify the search to explore other regions. The algorithm accomplishes this by making a series of random 2-interchange hops by the operations (2,0), (2,2), (2,1), and relinking. Relinking is a new operation introduced here to give more flexibility, and is illustrated in Fig. 2. The hop to a new solution is only granted if this move and resulting new S is not Tabu. At all times, a GB solution always overrides the Tabu restriction. At every successful random hop, the new solution is recorded in the candidate list and ranked for further local intensification later. After a number of iterations, the diversification process ends with a simple procedure that selects the least cost solution that is not a Tabu from the elite list and updates the current solution with this elite solution. Based on this new solution, a new round of intensification and diversification is triggered. The algorithm stops under one of the following conditions:

- a maximum number of iterations of the present solution has been reached;
- no feasible hops can be made.

To ensure that the GB solution is indeed the optimum in its neighbourhood, we apply 2-interchange (GB) at the end of the program. The complete strict Tabu search (S-TABU) algorithm follows.

Algorithm 3. Strict Tabu search

```
S-TABU1: Obtain an initial solution by PFIH and update the GB solution with the present solution;
S-TABU2: Initialise Tabu move list and Tabu solution list as well as candidate list; add current solution to the Tabu solution list;
S-TABU3: Do intensification, a 2-interchange (GB) procedure;
S-TABU4: Do diversification, and record random solutions encountered during random hops in the candidate list;
S-TABU5: If not diversified and total number of iterations is less than MAX_ITERATION,
    go to S-TABU3;
S-TABU6: Else improve the GB solution obtained so far by 2-interchange (GB) and return the improved solutions;
S-TABU7: Terminate S-TABU and print result.
```

7. Genetic algorithm

GA, originally developed by Holland [7], is an adaptive heuristic that simulates the optimisation process with the natural evolution of genes in a population of organisms as shown in Fig. 3. The GA maintains a population of candidate members over many generations. The population members are string entities of artificial chromosomes. Chromosomes are usually fixed length binary or integer strings. A special selection mechanism will pick up parents to go though crossover and mutation procedures and produce some children to replace themselves. A new generation is formed with all the parents replaced. The termination criterion of GA is convergence within a tolerable number of generations. In this paper, we

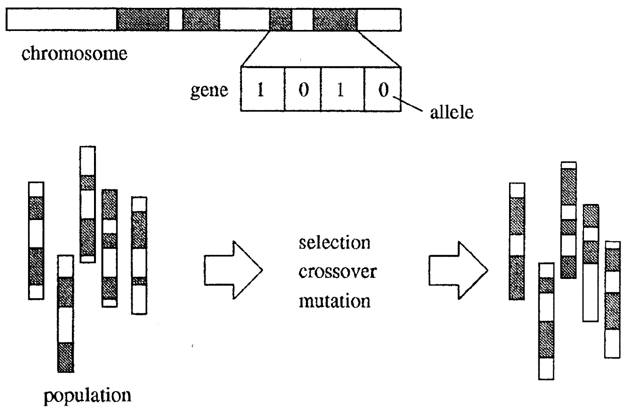

Fig. 3. The basic concept of genetic algorithm: a simple chromosome structure with binary genes. A population of chromosomes can be used to generate offspring undertaking selection, crossover and mutation operations.

combined a number of GA techniques such as new crossover operators and adaptive mutation probability, to solve VRPTW problems with satisfactory results.

7.1. Chromosome representation

Like in other GA applications, the members of a population in our GA for VRPTW are string entities of artificial chromosomes. The representation of a solution we use here is an integer string of length N, where N is the number of customers in question. Each gene in the string, or chromosome, is the integer node number assigned to that customer originally. The sequence of the genes in the chromosome is the order of visiting these customers. Using the same example, the chromosome string that represents the solution in Fig. 1 is now:

$$7 \to 8 \to 9 \to 11 \to 12 \to 2 \to 3 \to 1 \to 4 \to 5 \to 6 \to 10 \tag{18}$$

This representation is unique, and one chromosome can be only decoded to one solution. It is a 1-to-1 relation. Note we link the last customer visited in route i with the first customer visited in route $i+1$ to form one string of all the routes involved just like in the Tabu list, but we do not put any bit in the string to indicate the end of a route now, because such delimiters in a chromosome greatly restrain the validity of children produced by crossover operations later. To decode the chromosome into route configurations, we simply insert the gene values into new routes sequentially, similar to PFIH. There is a chance that we may not get back the original routes after decoding, but it is generally assumed that minimising the number of routes helps in minimising the total travel cost, therefore, packing a route to its maximum capability implies a potential good solution as a result.

7.2. Creation of initial population

Under the same assumption that we have made in the last sections, i.e. the solution from PFIH is reasonably good and in the vicinity of the GB solution, we create an initial population in relation to this solution. The way to do that is by letting the PFIH solution S_0 and its random neighbours $\forall S \in N_\lambda(S_0)$ describe a portion of the starting population. The rest of the population is generated on a totally random basis, unrelated to S_0. The reason for having this mixed population is that: a population of members entirely from the same neighbourhood cannot go too far from there and hence give up the opportunity to explore other regions. The proportion of relevant chromosomes and random chromosomes is governed by a parameter RAND_RATIO. The higher this ratio, the more diverse the initial population. This parameter also reflects the confidence level of the user to the PFIH solution. If there is a high chance that global optimum is located in $N_\lambda(S_0)$, then it is undoubtedly economical to have a small RAND_RATIO so that the population converges to the optimum sooner. The total population size is set at 100 and the number of generations ranges between 500 and 1000, a compromise between computation time and final result. Certainly, the more generations the program runs, the more optimised the solution in most cases, unless the optimum has already been reached.

7.3. Selection

After we have a population of candidates, we need some mechanism to select parents for mating and reproduction. A tournament selection mechanism is used for this purpose. In tournament selection, two identical copies of the population of size N are maintained at every generation. In the beginning, both populations are arbitrarily ranked. For population P_1, each pair of adjacent chromosomes (with indices $2i$ and $2i+1$) in the population are compared. The one with

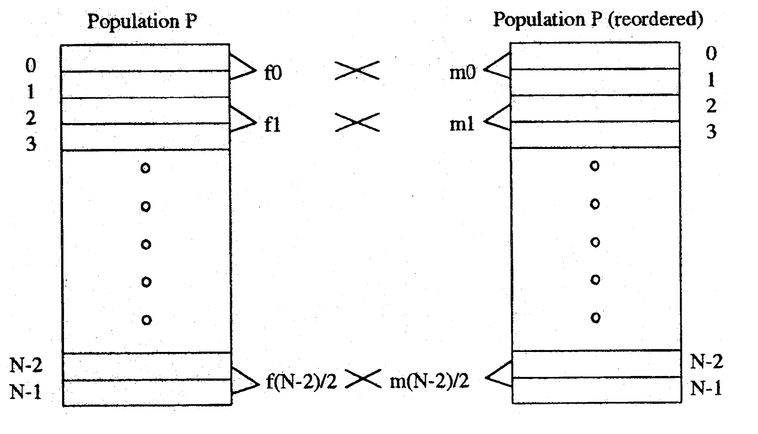

Fig. 4. Tournament selection. Tournament selection procedure: the first time selection will choose $N/2$ parents, after reordering the population, the second time selection will choose the other $N/2$ parents.

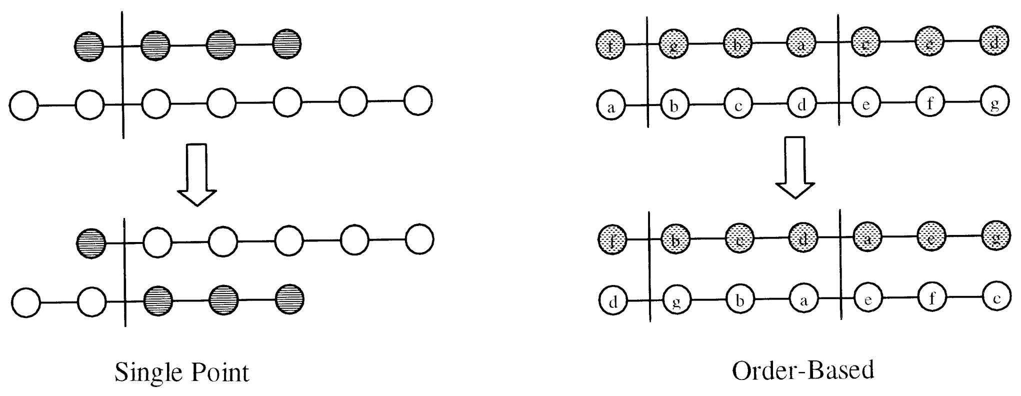

Fig. 5. Simple crossover vs. ordered crossover: demonstration of single point and order-based crossover operations. Single point crossover needs only one cut point for each chromosome. On the other hand, order-based crossover needs two cut points and repair of chromosomes must be performed after the operation.

smaller fitness value qualifies to be a potential parent and let us call it f_i. After comparing all the pairs in P_1, we have $N/2$ 'fathers', namely $f_0, f_1, \ldots, f_{(N-2)/2}$. Repeat this process for population P_2, and we get $m_0, m_1, \ldots, m_{(N-2)/2}$, a set of 'mothers' as well. Subsequently, f_i and m_i are mated, and totally there are N chromosomes chosen. The procedure is graphically illustrated in Fig. 4. The implication in this selection scheme is that genetically superior chromosomes are given priority in mating but average entities have some chance of being selected too, provided they happen to be compared with 'worse-off' chromosomes.

7.4. Reproduction

Reproduction is one of the most crucial functions in the chain of biological evolution. Similarly, reproduction in GA serves the important purpose of combining the useful traits from parent chromosomes and passing them on to the offsprings. An efficient and smart reproduction mechanism is largely responsible for high GA performance. The reproduction of GA consists of two kinds of operations, crossover and mutation.

Conventional single/double point crossover operations are relevant to string entities that are orderless, or of different length. They put two integer/binary strings side by side and make a cut point (or two cut points) on both of them. A crossover is then completed by swapping the portions after the cut point (or between two cut points) in both strings (see Fig. 5). In the context of VRPTW, where each integer gene appears only once in any chromosome, such a simple procedure unavoidably produces invalid offspring that have duplicated genes in one string. To prevent such invalid offspring from being reproduced, we define a set of order-based crossover operators below.

The PMX crossover [12]. The permutation crossover (PMX) method proceeds by choosing two cut points at random, e.g.

Parent 1: h k c e f d   b l a   i g j
Parent 2: a b c d e f   g h i   j k l

The cut-out section defines a series of swapping operations to be performed on the second parent. In the example case, we swap b with g, l with h and a with i, and end up with the following offspring:

Offspring: i g c d e f   b l a   j k h

Performing similar swapping on the first parent gives the other offspring:

Offspring: l k c e f d   g h i   a b j

Heuristic crossover. A random cut is made on two chromosomes. From the node after the cut point, the shorter between the two edges leaving the node is chosen. The process is continued until all positions in the chromosomes have been considered. Suppose, we have the following parents:

Parent 1: h k c e f d   b l a i g j
Parent 2: a b c d e f   g h i j k l

Assume we choose b to be the first gene in the new chromosome, we have to first swap b and g in parent 2. After swapping, if the distance $d_{\mathrm{bl}} > d_{\mathrm{bh}}$, where d_{bl} is the distance between node b and node l in parent 1 and d_{bh} is the distance between node b and node h in parent 2, we then choose h to be the next node and swap l and h in the first parent or delete h in the first parent to avoid duplication later. This process is continued until a new chromosome of the same length and comprising all the 12 alphabets are formed. Note that the result varies depending on whether swapping or deletion is undertaken. We named heuristic crossover with swapping *HeuristicCrossover*1 and *Heuristic Crossover*2 otherwise.

Merge crossover. Unlike heuristic crossover, which rearranges the parents according to distance to produce children, merge crossover operated on the basis of a predefined time

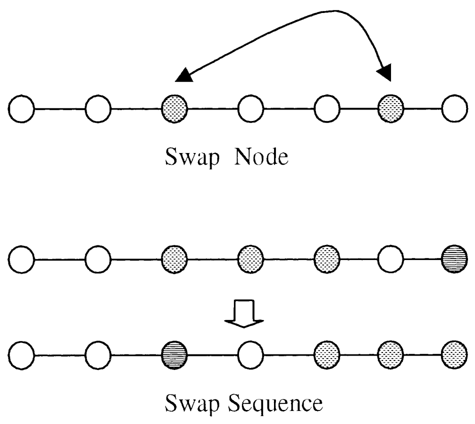

Fig. 6. Several common mutation operations: demonstration of swap node and swap sequence mutation operations. Unlike swap node operation, swap sequence operation involves reversing sequence of some nodes.

precedence. This time precedence is often summarised from the time windows imposed by each node. The author created such precedence from the latest arrival time of each node. In the same example:

Parent 1: h k c e f d b l a i g j
Parent 2: a b c d e f g h i j k l

Similar to heuristic crossover, a random cut point is selected and a first gene b is chosen randomly from the first parents and swapping is done to second parent. The node, which comes earlier in the time precedent becomes the next gene in the new chromosome. Here, we define two kinds of merge crossover just like the heuristic crossover case, *MergeCrossover*1 and *Merge-Crossover*2, to differentiate the swapping and deletion schemes.

It is noted that both heuristic crossover and merge crossover produce only one offspring from a pair of parents. Inspired by the fact that both geographic locations and time sequences are important in vehicle routing, we decided to combine the two crossover operators so that two parents now produce two children, each of which comes from either the heuristic crossover or the merge crossover. Naturally, we thus have four combined operators from the four combinations, namely H1M1, H1M2, H2M1 and H2M2. Experimental results on the total five crossover operators including PMX indicated that H1M2 outperforms the rest especially in clustered data sets. Not every pair of parents ought to reproduce in every generation. How many parents are to crossover is governed by the *probability of crossover*, a fix real number between 0 and 1. In our GA, we set it at 0.77, a moderate value usually used in other GA implementation as well. When a couple of parents are determined not to crossover, they are copied verbatim to the next generation.

Mutation is a complementary operation to crossover. The main purpose of mutation is to avoid over homogeneous population by bring random, unrelated traits into the present population and increase the variance of the population. Several mutation operators have been proposed in the literature and they are shown in Fig. 6. Because the chromosomes are of fixed length in our implementation, only swap node and swap sequence are used here. There is another important parameter associated with the mutation operations. This is the probability of mutation, or P_{mutation}, taking on values from [0,1]. Earlier research has shown that excessive P_{mutation} values drives the GA into convergence sooner than necessary, often resulting in undesirable local optimal solutions; small P_{mutation} produces the opposite result: intolerably slow convergence.

In this project, the author developed a unique *adaptive mutation probability scheme*. The scheme adapts P_{mutation} to the standard deviation of the population:

$$S = \sqrt{\frac{\sum_{i=0}^{N-1} (x_i - \bar{x})^2}{N-1}} \tag{19}$$

where N is the population size, x_i the fitness value of individual i and $\bar{x}$ is the average fitness of the population. If S is greater or equal to a threshold value MINPOPDEV = 5, a minimum $P_{\text{mutation}} = P_{\min} = 0.06$ is used, otherwise $P_{\text{mutation}} = P_{\min} + 0.1 \times (\text{MINPOPDEV} - S)$. Hence, we can see a minimum mutation probability of 0.06 is always guaranteed.

7.5. Further improvements

Hill-climbing is a supplementary measure that takes advantage of local greedy search to improve the chromosomes produced from crossover and mutation procedures. In hill-climbing, a portion of the population are randomly selected and decoded into their respective solution form. These solutions then undergo a few iterations of removal and re-insertion operations and are eventually updated with the new, improved solutions. Note this is not a complete λ-interchange procedure due to the moderately large time requirement by each λ-interchange. Furthermore, to reduce the complexity and to prevent from over-reliance on good solutions, the probability of a chromosome to be selected for hill-climbing is only set at 0.5.

However, even after hill-climbing, there is still the possibility of degradation of the entire new generation. To restore some of the good chromosomes in the parent generation, the worst 4% chromosomes in the child generation is substituted with the best 4% in the parents. Note that the percentage of recovery should be less than the mutation rate at any time to have some mutated chromosomes escape the recovery process and bring the population out of a premature convergence.

Table 1
A snap comparison of the heuristics (the number on top of each cell is the average cost; the number on the bottom is the average CPU time (s) associated with the type of problem)

	2-INT	SA	Tabu	GA
C1	965	943	874	872
	25	84	557	556
C2	780	766	644	641
	55	166	1885	1073
R1	1469	1422	1292	1333
	46	78	1076	507
R2	1330	1279	1097	1124
	98	217	3323	851
RC1	1680	1657	1471	1547
	49	63	1016	432
RC2	1700	1642	1331	1343
	60	146	3217	835

With all the strategies we have described in the last few sections, finally the GA is stated as such:
Algorithm 4. Genetic algorithm

```
GA-1: Generate initial population of $N$ chromosomes (partly from PFIH and its mutation and partly from totally random selection).
GA-2: Evaluate the fitness value in terms of the objective function for each chromosome $x$ in the population, calculate average fitness and standard deviation, thus set mutation probability;
GA-3: Create a new population by repeating following steps until the new population is complete;
    1. [Selection] Select two parent chromosomes from a population by tournament selection;
    2. [Crossover] With a crossover probability cross over the parents to form a new off-spring (H1M2). If no crossover was performed, offspring is an exact copy of parents;
    3. [Mutation] With a mutation probability mutate new offspring at a random locus (swap node or swap sequence);
    4. [Accepting] Place new offspring in a new population;
    5. [Hill-climbing] Performing hill-climbing with 1-neighbourhood FB search;
    6. [Recovery] Replace the worse 4% chromosomes in the new population with the best 4% in the parents population;
GA-4: Update the old population with the newly generated population;
GA-5: If the certain number of generation is reached, stop, perform a 2-interchange (GB) on the best solution in the current population and return the improved solution;
GA-6: Else go to step 2.
```

8. Computation results and comparisons

We conducted most of the tests on a Pentium II 266 MMX industrial Personal Computer with 32M RAM. Because of the varying nature of the algorithms developed, the duration of tests is also varying. For λ-interchange LSD (GB), the algorithm goes up to 20 iterations, but may terminate earlier if the local optimum is encountered. We did not specify the number iterations for the SA to run, instead we specify the number of resets that can be made before the program terminates. The parameter for SA is set as follows: $T_s = 100$, $T_f = 0.001$, $\tau = 0.5$, $R = 3$ (number of resets). Under such settings, the effective number of iterations for SA ranges from 500 to 700, depending on the input. Our S-TABU program has to undergo at least 500 iterations before it terminates. Because complexity arises from Tabu list operations and the more extensive search that TS guarantees, S-TABU takes the longest time to complete on average. Finally, both of the GA implementations, GA-PMX and GA-H1M2 are tested, with a population of 100 and 1000 generations.

All four heuristics were tested with 56 Solomon's VRPTW instances [18] which are 100-customer problem sets. The 56 problems are categorised into six classes, namely C1, C2, R1, R2, RC1 and RC2. Problems which fall into C categories are clustered data, meaning nodes are clustered either geographically or in terms of time

Table 2
Relative average cost for our heuristics against the best known (this table compares the minimal costs between the published best solutions and our solutions. The percentages are derived from Table 1. The last column shows the number of new best solution obtained against the number of problems in each category)

Problems	Published best	2-INT (%)	SA (%)	Tabu (%)	GA (%)	No. new best
C1	827.54	+15.9	+13.6	+4.3	+3.9	6/9
C2	589.4	+32.3	+28.5	+5.6	+4.7	5/8
R1	1191.6	+23.2	+19.2	+5.8	+9.9	0/12
R2	1024.7	+29.5	+24.8	+3.3	+6.1	5/11
RC1	1361.3	+23.2	+21.1	+6.0	+10.7	1/8
RC2	1156.8	+47.0	+41.9	+11.8	+10.9	1/8

windows. Problems from R categories are uniformly distributed data and those from RC categories are hybrid problems that have the features of both C and R categories. In addition, C1, R1 and RC1 problem sets have narrower time window for the depot, whereas the other problem sets have wider time window for the depot. A snap comparison of four of the heuristics is tabulated in Table 1, which shows the average costs obtained by each heuristic method on all six categories and average CPU time.

This table is derived from the data of Appendix A. For example, in class C1 of heuristic 2-INT, the average cost 965 is obtained by averaging the nine instances in class C1. The average cost indicates the performance of each heuristic on different categories of instances. Table 1 compares the average costs and average CPU time for each of the algorithms implemented in all six categories of problems. In general, TS is the most effective heuristic, solving the largest number of benchmark problems to near optima and achieving the least average total cost in almost all categories. However, the computation time is about 2–3 times that of GA and almost 20 times that of SA. GA almost hit as many good results as TS but did not as well as TS in R1 and R2 problems. Nevertheless, GA still remains a strong competitor to TS as it is a good compromise of quality and time. SA is very fast and offers reasonably good solutions. The correlation between the number of vehicles required and the total cost incurred in a solution is not straightforward. Fewer number of routes can incur more cost, especially in homogeneous data (R class problems). R109, R201, R202 and R203 are examples of such instances.

Table 2 shows the percentage increase of average total distance of our solutions against the best-known solutions in the literature, as well as the number of solutions we have obtained that are better than or equivalent to the best solutions in the literature. The published best-known solutions are not obtained by one or a particular class of methods, therefore, the table demonstrates the position of our algorithms in all VRPTW methods. Our TS and GA are already close to the best ways of solving VRPTW. However, we can also deduce that there is thus far no single heuristic that is generic enough to solve problems of all situations, instead they are inevitably problem specific.

A detailed best solution (number of vehicles needed and total cost) of our heuristics against the best-known results to the author is presented in Appendix A. We obtained 18 new best-known results by the new heuristics. Our algorithms, especially TS and GA did very well with clustered problem sets C1 and C2. Many of the results obtained for these two classes are already optimal, proved by Lagrangian relaxation [9] and other mathematical programming methods. However, our methods are less successful with homogeneously distributed data.

9. Conclusion

The implementations of λ-interchange is a basic cornerstone of all the more complex heuristic algorithms. It clearly defines the meaning of λ-neighbourhood and the operators to explore such a neighbourhood. The method is simple and straightforward but useful in almost all kinds of local search procedures. Theoretically, a sequence of 2-interchange operations is able to bring the current solution to anywhere in the solution space.

We next studied SA and TS, two of the most talked-about combinatorial optimisation strategies in the 1990s. Our implementation of SA takes a systematic approach in local search. Our experiments show that such systematic search is more efficient than ‘random walk’. SA is a good compromise of speed and performance. Our strict Tabu search (S-TABU) algorithm is one of the many ways to employ the sophisticated memory-based meta-heuristic TS. S-TABU was able to solve many of the Solomon problem sets to near optima in an average of 1500 s on a Pentium II 266 MMX PC. This method apparently benefited from the use of long-term memory and diversification strategy. However, our S-TABU still lacks some of the important features of TS, such as influence. The possibility of further improvement is almost certain.

This paper has also implemented a new application of GA to VRPTW. Previously, Thangiah used cluster-first–route-second method when applying GA to VRPTW. In his GIDEON system [22], the angular differences were coded in the chromosomes. GA was only used to sector the customers within clusters. Other heuristics like 2-opt and SA had to be used to assist GA in routing the customers within one cluster. Strictly speaking, it is only a hybrid heuristic that constitutes some GA element. Our current implementation, however, is a complete GA with a more intuitive representation, a complete set of improvement schemes of many flexible parameters. The results from this GA system prove superior than that from the GIDEON system in all 56 Solomon instances. Other authors, like Blanton and Wainwright [1], applied GA with similar representation to VRPs, but almost all their work concentrates on the TSP, which involves only one route and has been proved NP-complete. Moreover, these authors failed to present their result data so no comparison can be made. The paper is thus among the first to solve complex VRPTW by GA and to present comprehensive results to all 56 100-customer benchmark problems.

Acknowledgements

The authors wish to thank the editor J.C. Kunz and the anonymous reviewers for their valuable comments and helpful suggestions, which greatly improved the paper.

Appendix A. Comparison of our best results with the historical best[1] (NV: number of vehicles; TD: total distance)

Problems	Published best		2-INT		SA		Tabu		GA	
	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD
C101	10	829	10	*828.937*	10	*828.937*	10	*828.937*	10	*828.937*
C102	10	827	10	923.375	10	923.375	10	901.527	10	868.798
C103	10	828.06	10	994.87	10	994.87	10	954.718	11	939.456
C104	10	824.78	11	1130.85	11	1130.85	10	895.774	10	963.72
C105	10	829	10	*828.937*	10	*828.937*	10	*828.937*	10	*828.937*
C106	10	827	10	1052.07	10	1052.07	10	941.154	10	*828.937*
C107	10	829	10	875.62	10	867.234	10	*828.937*	10	*828.937*
C108	10	827	10	878.089	10	876.427	10	*828.937*	10	*828.937*
C109	10	829	10	1100.71	10	1124.44	10	*828.937*	10	*828.937*
C201	3	590	3	*591.557*	3	*591.557*	3	*591.557*	3	*591.557*
C202	3	590	4	801.281	4	787.856	4	745.99	4	683.864
C203	3	591.55	4	1225.1	4	1208.94	4	727.221	4	745.934
C204	3	590.6	3	661.213	3	642.691	3	*590.599*	3	604.998
C205	3	589	3	625.333	3	613.327	3	*588.876*	3	*588.876*
C206	3	588	3	704.162	3	694.25	3	*588.493*	3	*588.493*
C207	3	588	3	725.404	3	704.365	3	600.841	3	593.195
C208	3	588	3	902.597	3	888.685	3	645.206	3	*590.873*
R101	18	1608	20	1847.37	20	1847.37	20	1707.95	20	1676.86
R102	17	1434	19	1720.46	18	1544.82	16	1488.59	18	1558.59
R103	13	1207	17	1551.34	17	1482.39	15	1293.85	15	1311.81
R104	10	982.01	12	1184.38	12	1184.38	11	1057.02	12	1128.29
R105	14	1377.11	17	1671.94	17	1595.68	16	1431.56	17	1496.37
R106	12	1252.03	14	1439	14	1434.3	14	1331.5	14	1357.19
R107	11	1126.69	13	1390.55	12	1270.04	12	1174.89	13	1240.82
R108	10	968.59	12	1256.61	12	1186.34	11	1039.34	12	1091.69
R109	11	1205	14	1525.45	14	1515.38	14	1256.36	15	1300.29
R110	11	1080.36	13	1444.63	13	1430.35	13	1179	13	1315.56
R111	10	1104.83	13	1371.64	13	1370.21	13	1148	12	1202.31
R112	10	953.63	12	1215.27	12	1180.18	11	1088.32	12	1097.64
R201	4	1354	5	1791.42	5	1726.13	5	1437.49	8	*1329.74*
R202	3	1530.49	4	1610.02	4	1581.64	5	*1272.6*	7	*1307.03*
R203	3	1126	4	1475.63	4	1248.55	4	*1081.04*	6	*1086.43*
R204	2	914	3	1098.67	3	1088.06	3	*895.867*	6	956.384
R205	3	1128	4	1370.68	4	1344.2	4	1150.34	5	*1131.18*
R206	3	833	3	1341.22	3	1369.5	4	1103.22	5	1187.25
R207	3	904	3	1167.11	3	1153.65	3	1007.3	4	1016.63
R208	2	759.21	3	988.367	3	971.572	3	806.797	3	845.937
R209	2	855	4	1210.96	4	1206.58	4	1110.3	5	1097.42
R210	3	1052	4	1312.91	4	1238.14	4	1071.3	6	1136.54
R211	3	816	3	1232.48	3	1140.65	3	946.354	7	932.483
RC101	14	1669	17	1948.94	17	1940.57	16	1734.17	17	1728.3
RC102	12	1554.75	16	1803.95	16	1777.02	14	1562.62	17	1603.53
RC103	11	1110	14	1627.02	14	1620.35	13	1377.93	14	1519.83
RC104	10	1135.83	12	1408.5	12	1408.5	11	1259.28	12	1276.02
RC105	14	1602	17	1920.19	17	1809.78	16	*1597.67*	17	1688.77
RC106	11	1448.26	14	1656.29	14	1645.24	14	1476.15	14	1491.58
RC107	11	1230.54	15	1677.92	15	1653.65	13	1392.97	14	1462.3
RC108	10	1139.82	12	1377.93	13	1335.06	12	1264.5	12	1333.15
RC201	4	1249	5	2070.4	5	1891.9	5	1617.5	10	1565.67
RC202	4	1221	5	1970.66	5	1956.97	5	1429.04	10	1353.27
RC203	3	1203	4	1627.74	4	1522.68	4	*1179.67*	6	*1189.06*
RC204	3	879	4	1314.21	4	1290.89	4	939.678	4	989.943
RC205	4	1389	5	1923.9	5	1907	5	1487.49	9	1465.8
RC206	3	1213	4	1663.87	4	1645.17	4	1357.32	5	1388.13
RC207	3	1181	4	1539.85	4	1497.54	4	1295.9	6	1304.48
RC208	3	919	3	1490.27	3	1422.94	3	1040.47	6	1003.43

[1] The best-known solutions were obtained from the following sources: [9,17,21,23] and http://dmawww.epfl.ch/~rochat data/solomon.html.

References

[1] Blanton Jr JL, Wainwright RL. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms, 1993.

[2] Eilon S, Watson-Grandy C, Christofides N. Distribution management: mathematical modeling and practical analysis. New York: Hafner, 1971.

[3] Fisher ML, Jornsten KO, Madsen OBG. Vehicle routing with time windows. Working Paper, 1992.
[4] Fisher ML. Vehicle routing, Handbooks in operations research and management science, vol. 8. Amsterdam, New York: Elsevier, 1995
[5] Gendreau M, Hertz A, Laporte G. A Tabu search heuristic for the vehicle routing problem. Mgmt Sci 1994;40:1276–90.
[6] Glover F. Manuel laguna, Tabu search. Dordrecht: Kluwer Academic Publishers, 1997.
[7] Holland JH. Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press, 1975.
[8] Kirkpatrick S, Gelatt Jr. CD, Vecchi MP. Optimization by simulated annealing. Science 1983;20:671–80.
[9] Kohl N, Madson O. An optimization algorithm for the vehicle routing problem with time windows based on Lagrangian relaxation. Working Paper, 1995.
[10] Kolen AWJ, Kan A.H.G.R., Trienekens HWJM. Vehicle routing with time windows Oper Res 1987;35:266–274.
[11] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equations of state calculations by fast computing machines. J Chem Phys 1953;21:1087–92.
[12] Oliver IM, Smith DJ, Holland JRC. A study of permutation crossover operations on the traveling salesman problem. In: Proceedings of the Fourth International Conference on Genetic Algorithms, 1991.
[13] Osman IH, Christofides N. Simulated annealing and descent algorithms for capacitated clustering problem. Research Report. Imperial College, University of London, 1989.
[14] Osman IH. Meta-strategy simulated annealing and Tabu search algorithms for the vehicle routing problem. Annu Oper Res 1993;41:777–86.
[15] Potvin JY, Kervahut T, Garcia BL, Rousseau JM. The vehicle routing problem with time windows. Part I. Tabu search. INFORMS J Comput 1996;8:158–64.
[16] Prinetto P, Rebaudengo M, Sonza Reorda M. Hybrid genetic algorithms for the traveling salesman problem. In: Proceedings of the Fifth International Conference on Genetic Algorithms, 1993.
[17] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. Working Paper, 1998.
[18] Solomon MM. Algorithms for vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35(2):254–66.
[19] Taillard E. Parallel iterative search methods for vehicle routing problems. Networks 1993;23:661–73.
[20] Thangiah SR, Osman IH, Sun T. Algorithms for the vehicle routing problems with time deadlines. Working Paper, 1992.
[21] Thangiah SR, Osman IH, Sun T. Hybrid genetic algorithm, simulated annealing and Tabu search methods for vehicle routing problems with time windows. Technical Report SRU-CpSc-TR-94-27. Computer Science Department, Slippery Rock University, 1994.
[22] Thangiah SR. An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In: Proceedings of the Sixth International Conference on Genetic Algorithms, 1995.
[23] Thangiah SR. Vehicle routing with time windows using genetic algorithms. Working Paper, 1995.
[24] Willard JAG. Vehicle routing using R-optimal Tabu search. MSc Thesis. London: Management School, Imperial College, 1989.
[25] Desrochers M, Desrosier J, Solomon M. A new optimization algorithm for vehicle routing problems with time windows. Oper Res 1992;40:342–355