Heuristics for Vehicle Routing Problem with Time Windows

K. Q. Zhu*, K. C. Tan†, L. H. Lee‡

December 10, 1999

Abstract

This paper documents our investigation into various heuristic methods to solve Vehicle Routing Problem with Time Windows (VRPTW) to optimal or near optimal solutions. The objective of the problem is to serve a number of customers within predefined time windows at minimum cost, without violating the capacity and total trip time constraints for each vehicle. The heuristics we are exploring here are mainly third generation Artificial Intelligent (AI) algorithms such as λ-Interchange, Simulated Annealing (SA), and Tabu Search (TS). Based on the original SA theory proposed by Kirkpatrick and the work by Thangiah, we update the cooling scheme and develop a fast 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. We applied each of the heuristics developed to Solomon's 56 VRPTW 100-customer instances, and yielded 17 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 three advanced search methods for VRPTW, together with their comprehensive results.

Keywords: *vehicle routing problem*, *time windows*, *combinatorial optimization*, *heuristics*, *simulated annealing*, *tabu search*.

1 Introduction

The Vehicle Routing Problem with Time Windows is given by a set of identical vehicles V, a special node called the depot, a set of customers C to be visited, a directed network connecting the depot and the customers. Let us assume there are K vehicles, $V=\{0, 1, 2, \ldots, K-1\}$, and $N+1$ customers, $C=\{0, 1, 2, \ldots, N\}$. For simplicity, we denote depot as customer 0. Each arc in the network corresponds to a connection between two nodes. A route is defined as starting from the depot, going through a number of customers and ending at the depot. The number of routes in the traffic network is equal to the number of vehicles used, that is, K. Therefore, exactly K directed arcs leave the depot and K arcs return to the depot. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network. Every customer in the network must be visited only once by one of the vehicles. Since each vehicle has a limited capacity q_k (identical for all vehicles in the question), and each customer has a varying demand m_i, q_k must be greater than or equal to the summation of all demands on the route traveled by that vehicle k. On the other hand, any customer must be serviced within a pre-defined time interval, limited by an earliest arrival time and latest arrival time. Vehicles arriving later than the latest arrival time are penalized while those arriving earlier than the earliest arrival time incur waiting. Vehicles are also supposed to complete their individual routes within a total route time which is essentially the time

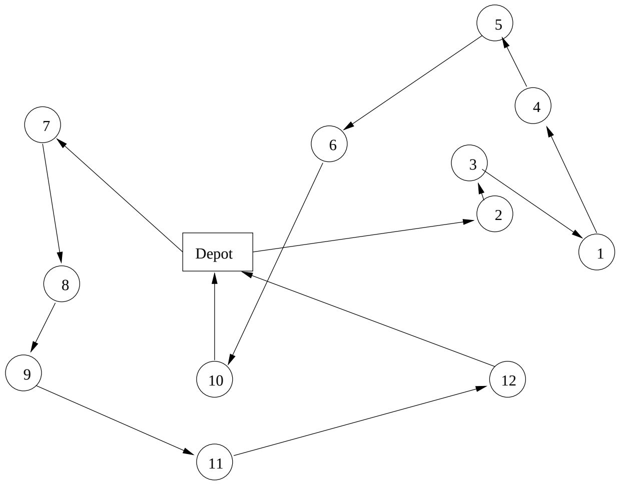

Figure 1: A Vehicle Routing Problem

*Kenny Qili Zhu, Dept. CS, National University of Singapore. Email: kzhu@comp.nus.edu.sg

†Kay Chen Tan, Dept. EE, National University of Singapore. Email: eletankc@nus.edu.sg

‡Loo Hay Lee, Dept. ISE, National University of Singapore. Email: iseleelh@nus.edu.sg

window of the depot.

2 An Initial Solution

Most heuristic search strategies involves finding an initial feasible solution and then improving on that solution using local or global optimization techniques. Here we make use of *Push-Forward Insertion Heuristic (PFIH)*, first introduced by Solomon[13] in 1987 as a method to create initial route configuration. The PFIH starts a new route by selecting an initial customer and then inserting customers into the current route until either the capacity of the vehicle is exceeded or it is not feasible to insert another customer into this route in terms of time. The cost function for inserting a customer as the first node C_1 into a new route is as follows:

$$\text{Cost}\,(C_1) = -\alpha t_{oi} + \beta l_i + \gamma((\frac{|p_i - p_j|}{360})t_{0i}) \quad (1)$$

where p_i is the polar angle of the customer considered and p_j is the polar angle of the last visited customer in the last formed route. The unrouted customer with the lowest cost is selected as the first customer to be visited. The polar coordinate angle of the customer with respect to the depot in (1) is normalized in terms of the distance. This normalization allows comparison of the distance, latest deadline and angular value of the customer in common units. The weights for the three criteria were derived empirically and were set as $\alpha = 0.7$, β= 0.2, γ= 0.1. Once the first customer is selected for the current route, the heuristic selects from the set of unrouted customers the customer j^* which minimizes the total insertion cost between every edge $\{k,\ l\}$ in the current route without violating the time and capacity constraints. The insertion cost of other nodes in a route, as opposed to Eq. (1) is given by

$$\text{Insertion cost of } C_i = D_k + \phi W_k + \eta O_k + \kappa T_k \quad (2)$$

ϕ was set to 1% of the total distance D_k. When calculating the penalty weight factors η and κ in (2), η was set to 10% of D_k and κ to 1% of D_k. The penalty factors were chosen in this manner to allow penalty relative to the total distance traveled by the vehicle. The customer j^* is inserted in the least cost position between $\{k^*\ ,\ l^*\}$ in the current route if it is valid, and the selection process is repeated until no further customers can be inserted. At this stage, a new route is created and the above is repeated until all customers are routed. The actual number of vehicle deployed, K, is determined by the heuristic, and it provides an upper bound for the number of routes in the solution.

3 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 neighborhood is searched. Introduced by Osman & Christofides[11], a λ-*interchange* generation mechanism is based on customer interchange between sets of vehicle routes.

Given a feasible solution for the VRPTW represented by $S = \{R_1, ..., R_p, ..., R_q, ..., R_k\}$, where R_p is a set of customers serviced by a vehicle route p. A λ-interchange between a pair of routes R_p and R_q is a replacement of subset $S_1 \subseteq R_p$ of size $|S_1| \le \lambda$ by another subset $S_2 \subseteq R_q$ of size $|S_2| \le \lambda$, to get the two new route sets $R'_p = (R_p - S_1) \cup S_2$ and $R'_q = (R_q - S_2) \cup S_1$, and a new neighboring solution $S' = \{R_1, ..., R'_p, ..., R'_q, ..., R_k\}$. The neighborhood $N_\lambda(S)$ of a given solution S is the set of all neighbors $\{S'\}$ generated by the λ-interchange method for a given λ.

Let the permutation σ be the order of vehicle indices in a given solution $S = \{R_1, ..., R_p, ..., R_q, ..., R_k\}$ (e.g. $\sigma(p) = p$, $\forall p \in [0,\ K\text{-}1]$). An ordered search selects all possible combination of pairs $(R_p,\ R_q)$ according to the following order without repetition: $(R_{\sigma(1)},\ R_{\sigma(2)})$, ..., $(R_{\sigma(1)},\ R_{\sigma(K)})$, $(R_{\sigma(2)},\ R_{\sigma(3)})$, ..., $(R_{\sigma(K-1)},\ R_{\sigma(K)})$. A total number $C_{k,2} = \frac{K\times(K-1)}{2}$ of different pair of routes (R_p, R_q) are examined to define a cycle of search .

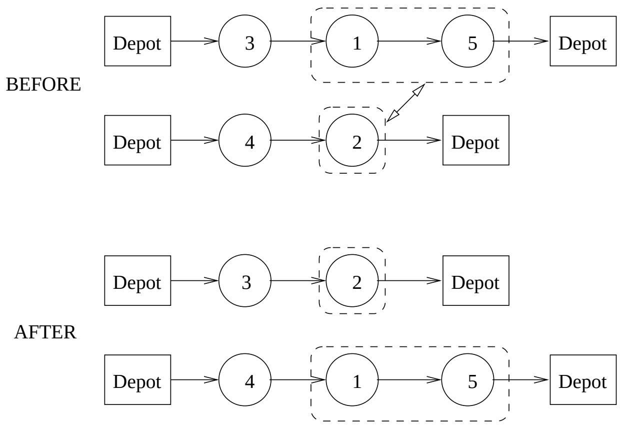

Figure 2: An Operation (2, 1)

For heuristics based on a descent algorithm the same permutation σ is used after each cycle of search is completed. We consider the case of λ = 1 and λ = 2 for the neighboring search. The λ-interchange method defines a series of insertion and exchange operators. The operator (0, 1) on routes $(R_p,\ R_q)$ indicates a shift of one customer from route q to route p. The operators (1, 0), (2, 0) and (0, 2) indicate shifting of customers between two routes. The operator (1, 1) or (2, 2) on routes (R_p, R_q) indicates an exchange of one or two customers between route p and q, and so forth. Figure 2 illustrates one of the operations. Note we have defined that (2, 0) shifts two adjacent nodes in the route, and so on. In this way, all these operators have only four operants each: (R_p, i, R_q, j), where i and j are positions where nodes will be removed and re-inserted.

After a solution is generated a criterion is required for accepting or rejecting a move. The *First-Best (FB)* strategy will select the first solution in *S'* in $N_\lambda(S)$ in the neighborhood of S that results in a decrease in cost with respect to a cost function. The *Global-Best (GB)* strategy will search all solutions *S'* in $N_\lambda(S)$ in the neighborhood of S and select the one which will result in the maximum decrease in cost with respect to a given cost function. For FB strategy, the order of searching the 2-neighborhood is (0, 1), (1, 0), (1, 1), (0, 2), (2, 0), (2, 1), (1, 2) and (2, 2) on any given pairs to generate neighbors. The order does not matter in GB case. Global-Best usually achieves better results than First-Best because it keeps track of all the improving moves, but is more expensive computationally. On the other hand FB is a blind search that accepts the first best result. But that is not always the case. Figure 3 is such an exception. Following the principle of GB strategy, the algorithm will move to point B after a search of its 2-neighborhood, and will inevitably be trapped in a local optimum. FB strategy, however, may enable the search to jump to point A first and subsequently converge to a real optimum.

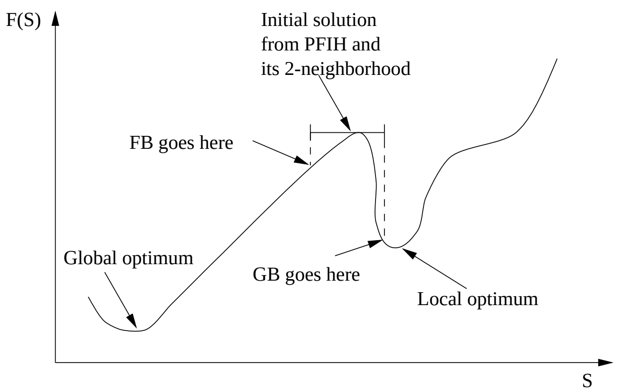

Figure 3: A Case where FB is Superior to GB

The λ-interchange Local Search Descent 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[1]. The LSD result is dependent on the initial feasible solution. In the worst case, LSD is only able to search thoroughly a 2-neighborhood of the initial solution, and if no better moves can be realized the algorithm will exit after a number of iterations.

4 Simulated Annealing

Simulated Annealing (SA) is a stochastic relaxation technique that finds its origin in statistical mechanics[10]. In annealing process, 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 analogy can be used in combinatorial optimizations 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].

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 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

$$\exp(-\frac{d}{kT}) \tag{3}$$

, where k is the *Boltzmann constant* and T is the absolute temperature at present. This basic step can be repeated indefinitely, which is known as a *metropolis step.* 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

$$\frac{\exp(-e_i/kT)}{\sum_j \exp(-e_j/kT)}. \tag{4}$$

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

[1] Iteration is defined as a valid change of one solution state to another .

being the current state. These probabilities are derived for a never ending executing of the metropolis loop.

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

$$T_k = \frac{T_{k-1}}{1 + \tau\sqrt{T_{k-1}}} \qquad (5)$$

where T_k is the current temperature at iteration k and τ is a small time constant. The square root of T_k is introduced in the denominator to speed up the cool process. Here we use a simple monotonously decreasing function to replace the $1/\log k$ scheme. It is found that our scheme, although converges sooner to zero, gives fairly good results in much less time. The algorithm attempts solutions in the neighborhood of the current solution randomly or systematically and calculates the probability of moving to those solutions according to:

$$P(\text{accepting a move}) = \exp(-\frac{\Delta}{T_k}). \qquad (6)$$

This is a modified version of Eq. (3), where $\Delta = C'(S) - C(S)$, $C(S)$ is the cost of the current solution and *C'(S)* is 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 is getting exponentially smaller. When the temperature has gone to the final temperature $T_f = 0.001$ or there is no more feasible moves in the neighborhood, we reset the temperature to

$$T_r = Max(T_r/2, T_b) \qquad (7)$$

where T_r is the reset temperature, and was originally set to T_s, and T_b is 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 neighborhood, the 2-interchange approach was adopted. And every time a global best solution is found, a 2-interchange (GB) procedure is executed to search for possible elite solutions around it. The procedure terminates after a number of resets.

In general, our SA implementation is a simple and fast algorithm that solves many VRPTWs to near optima. Due to the Global Best approach in local neighborhood 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.

5 Tabu Search

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. Our design of TS for VRPTW, known as Strict Tabu, or S-TABU, combines short term recency memory and long term frequency memory. We have created a special multi-functional list structure that serve both purposes.

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

$$\langle R_1, node1, position1, R_2, node2, position2\rangle \qquad (8)$$

where R_1 and R_2 are two routes under operation, *node1* is a node from R_1 and *position1* is the original position of node1. Likewise for *node2* and *position2*. Another recency information stored in Tabu List is the solution configurations. Every solution that has been recently encountered is coded into an integer string. If we have the following solution:

Route No. 1 is 0 -> 3 -> 2 -> 4 -> 5 -> 0

Route No. 2 is 0 -> 10 -> 6 -> 1 -> 12 -> 11 -> 0

Route No. 3 is 0 -> 9 -> 8 -> 7 -> 0

The coded integer string is then

$$\langle 3\,2\,4\,5\,0\,10\,6\,1\,12\,11\,0\,9\,8\,7\,0\rangle. \qquad (9)$$

The zeros are there to delimit the routes. The total cost of this solution is also attached to the string. The lifetime of the tabu moves and tabu solutions on the Tabu List is governed by the *Tabu List Size (TLS)*. In this paper, we fix the TLS at 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. High tabu-hit frequency implies the search has been caught in a local optimum and hence some action has to be taken.

Another list structure in S-TABU is *Candidate List*, which employs longer term memory. This list stores the *elite solutions* the system has discovered in the search process, also in integer string form like (9), but ranked according to the total cost attached to the string, instead of a 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. The size of the Candidate List can be variable, but is generally twice the size of Tabu List.

S-TABU begins with an initial solution obtained from PFIH. This solution is *intensified* by undergoing a 2-interchange (GB) 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 global best, it is also copied to the elite list for future exploration. The intensification terminates when the entire $N_2(S)$ has been searched without any improved solutions or the maximum tabu-hit frequency has been reached.

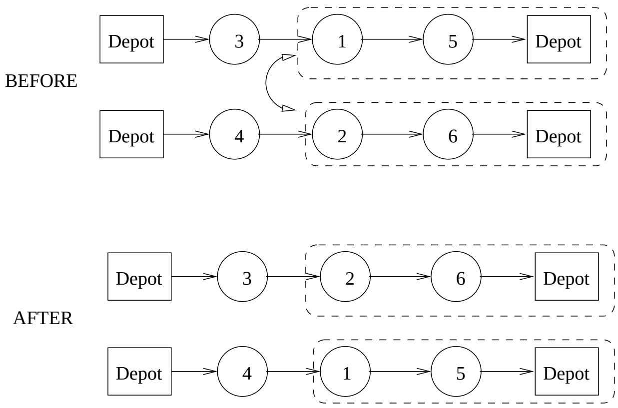

Figure 4: A Relink Operation

After one round of intensification, the current solution S is believed to have reached the optimum of its neighborhood. To *diversify* the solution, the algorithm then makes a series of random 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 Figure 4. The hop to a new solution is only granted if this move and resulting new S is not tabu. At all times, a global best solution always overrides the tabu restriction, which is the *aspiration criterion*. At every successful random hop, the new solution is recorded in the Candidate List and ranked for further local intensification. After a number of iterations, the diversification process ends with a simple procedure that selects the least cost solution which 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 if a total number of iteration of the present solution has been reached, or diversification failed as no feasible hops can be made. To ensure that the global best solution is indeed the optimum in its neighborhood, we apply 2-interchange (GB) at the end of the program.

6 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 of 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.

All three heuristics were tested with 56 Solomon's VRPTW instances[13] which are 100-customer problem sets. The 56 problems are categorized into 6 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 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 windows for the depot whereas the rest problem sets have wider time windows for the depot. A snap comparison of the heuristics is tabulated in Table 1, which shows the average costs obtained by each heuristic method on all 6 categories and average CPU times associated with the costs.

It compares the average total costs and average CPU time for each of the algorithms implemented in all 6 categories of problems. In general TS is the most effective heuristic, solving the largest number of benchmark problems to optima and achieving the least average total cost in almost all categories. However the computation time is about 20 times that of SA and

Methods	2-INT	SA	TABU
C1	965	943	874
	25	84	557
C2	780	766	644
	55	166	1885
R1	1469	1422	1292
	46	78	1076
R2	1330	1279	1097
	98	217	3323
RC1	1680	1657	1471
	49	63	1016
RC2	1700	1642	1331
	60	146	3217

Table 1: A Snap Comparison of the Heuristics
Note: The number at top of each cell is the average total cost; the number at the bottom is the average CPU time in seconds associated with the type of problem.

almost 40 times that of 2-INT. SA is very fast and offers reasonably good solutions. All three heuristics tend to give better results to clustered data. 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.

Prob.	2-INT	SA	TS	New Best
C1	+15.9%	+13.6%	+4.3%	6/9
C2	+32.3%	+28.5%	+5.6%	5/8
R1	+23.2%	+19.2%	+5.8%	0/12
R2	+29.5%	+24.8%	+3.3%	4/11
RC1	+23.2%	+21.1%	+6%	1/8
RC2	+47.0%	+41.9%	+11.8%	1/8

Table 2: Relative Average Cost for Our Heuristics Against the Best Known
Note: This table compares the minimal costs between the published best solutions and our average solutions. The last column shows the number of new best solution obtained against the number of problems in each category.

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. It is seen that our TS is already close to the best ways of solving VRPTW. However we can also deduce that there is thus far no single heuristic what is generic enough to solve problems of all types, instead they are inevitably problem specific. We have obtained 17 new best known results by the new heuristics. It can be seen that our algorithms, especially Tabu Search did very well with clustered problem sets C1 and C2. Many of the results obtained for these two classes are already optimal, proved by Lagrangean relaxation[9] and other mathematic programming methods. However, our methods are less successful with homogeneously distributed data. We believe that set partitioning algorithms or other clustering algorithms can be incorporated into our heuristics to improve the results.

7 Conclusion

The implementations of λ-interchange is a basic corner stone of the more complex heuristic algorithms later. It clearly defines the meaning of λ-neighborhood and the operators to explore such a neighborhood. The method is simple and straightforward but useful in almost all kinds of local search procedures. Thangiah has not been able to define clearly the operation (2, 0) and (2, 1) in his papers. And these points are clarified in Section 3 with figures and literal description. Our new definition is complete and easy to realize. Theoretically a sequence of 2-interchange operations are able to bring the current solution to anywhere in the solution space.

We next studied Simulated Annealing and Tabu Search, two of the most talked-about combinatorial optimization strategies in the 90's. Our implementation of SA takes a systematic approach in local search. It has been proved in the experiments 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 Tabu Search (TS). S-TABU was able to solve many of the Solomon problem sets to optima in an average of 1500 seconds on a Pentium II 266 MMX PC. This apparently benefited from the use of longer term memory and diversification strategy. Many authors have implemented their versions of TS as well, but none of them in the literature produced as many best solutions as we did at the time of writing. However, our S-TABU still lacks some of the important features of TS, such as influence. The possibility of further improvement is almost certain.

References

[1] Desrosier, J., Y. Dumas, M. Solomon and F. Soumis, *Time Constraint Routing and Scheduling*. Handbooks on Operations Research and management Science, Volume Network, North Holland, Amsterdam (1995).

[2] Eilon, S., C. Watson-Grandy and N. Christofides, *Distribution Management: Mathematical Modeling and practical Analysis*. Hafner, New York (1971).

[3] Fisher, M. L., K. O. Jornsten and O. B. G. Madsen, *Vehicle Routing with Time Windows*. Working paper (1992).

[4] Fisher, M. L. , *Vehicle Routing*. Handbooks in Operations Research & Management Science, Vol 8 (1995).

[5] Glover, F., *Tabu Search - Part I*. ORSA Journal on Computing (1989).

[6] Glover, F., Manuel Laguna, *Tabu Search*. Kluwer Academic Publishers (1997).

[7] Holland, J. H., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor(1975).

[8] Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, *Optimization by Simulated Annealing*. Science, 20:671-680 (1983)

[9] Kohl, N. and O. Madson, *An Optimization Algorithm for the Vehicle Routing Problem with Time Windows based on Lagrangean Relaxation*. Working paper (1995).

[10] Metropolis, N., A.W. Rosenbluth, M.N. Rosenbluth, A.H. *Teller and E. Teller, Equations of state calculations by fast computing machines*. The Journals of Chemical Physics, 21:1087-1092 (1953).

[11] Osman, Ibrahim H. and N. Christofides, *Simulated Annealing and Descent Algorithms for Capacitated Clustering Problem*. Research Report, Imperial College, University of London (1989).

[12] Osman, I. H., *Meta-strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem*. Annuals Operations Research 41 (1993).

[13] Solomon, M. M., *Algorithms for Vehicle Routing and Scheduling Problems with Time Window Constraints*. Operations Research 35 (2) (1987).

[14] Taillard, E., *Parallel Iterative Search Methods for Vehicle Routing Problems*. Networks 23, 661-673 (1993).

[15] Thangiah, S. R., I. H. Osman and T. Sun, *Algorithms for the Vehicle Routing Problems with Time Deadlines*. Working paper (1992).

[16] Thangiah, S. R., I. H. Osman and T. Sun, *Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows*. Working paper (1993).

[17] Thangiah, S. R. , *An Adaptive Clustering Method using a Geometric Shape for Vehicle Routing Problems with Time Windows*. Proceedings of the Sixth International Conference on Genetic Algorithms (1995)

[18] Thangiah, S. R., *Vehicle Routing with Time Windows using Genetic Algorithms*, working paper (1995).

[19] Willard, J. A. G., *Vehicle Routing using R-optimal Tabu Search*. MSc thesis, Management School, Imperial College, London (1989).